

# Visual Studio 2003 Visual J#

Copyright© 2016 Microsoft Corporation

El contenido de este documento se ha retirado y ya no se actualiza o admite. Algunos vínculos podrían no funcionar. El contenido retirado representa la versión más reciente actualizada de este contenido.

# Introducción a Visual J#

Esta sección proporciona información general sobre las capacidades, características y arquitectura del sistema de desarrollo de Visual J#.

## En esta sección

### [Información general sobre Visual J#](#)

Resume las ventajas, las características y las principales funciones de Visual J#.

### [Características de Visual J# Standard](#)

Describe las características incluidas en Visual J# .NET Standard.

### [Arquitectura de Visual J#](#)

Muestra la arquitectura del compilador de Visual J#, de las extensiones y del conversor binario en relación con las bibliotecas de JDK nivel 1.1.4, .NET Framework y Common Language Runtime.

### [Buscar información de referencia sobre Java y JDK](#)

Enumera fuentes de material de referencia sobre las bibliotecas de clases de Java y JDK nivel 1.1.4.

### [Buscar material de referencia en la documentación de Visual Studio .NET](#)

Enumera vínculos útiles a temas de la documentación de Visual Studio, Visual C#, Visual Basic y .NET Framework.

### [Información para los desarrolladores](#)

Breve introducción a Visual J#, sus aplicaciones y su relación con el lenguaje Java.

## Secciones relacionadas

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

# Información general sobre Visual J#

Visual J# es una herramienta que pueden utilizar los programadores de Java para generar aplicaciones y servicios que se ejecuten en .NET Framework.

Visual J# está orientado a Common Language Runtime y se puede utilizar para desarrollar aplicaciones .NET, incluidos Servicios Web XML y aplicaciones Web, de forma que se haga un uso total de .NET Framework. Las aplicaciones de Visual J# se benefician de:

- Integración entre lenguajes
- Seguridad mejorada
- Control de versiones e implementación
- Servicios de depuración y generación de perfiles

**Nota** Se puede utilizar Visual Studio® para depurar aplicaciones de Java, incluso si no se tiene Visual J# instalado en el equipo.

La extensión predeterminada de archivo de código fuente en Visual J# es .jsl.

Visual J# incluye:

- El compilador de Visual J#, que compila archivos de código fuente de Java como Lenguaje intermedio de Microsoft® (MSIL); vea [Opciones del compilador de Visual J#](#) para obtener más información.
- Un conversor binario que transforma código de bytes de Java en Lenguaje intermedio de Microsoft (MSIL); vea [Conversor binario de Visual J# .NET](#) para obtener más información.
- Las bibliotecas de clases desarrolladas de manera independiente y diseñadas para proporcionar la funcionalidad de la mayoría de las bibliotecas de clases de JDK nivel 1.1.4 y muchas de las clases del paquete **java.util** de JDK 1.2 especificadas en el currículo de becas avanzadas de informática (Advanced Placement curriculum for Computer Science) del sistema College Board en Estados Unidos. Vea [Compatibilidad con bibliotecas de clases](#) para obtener más información.
- Compatibilidad con Windows® Foundation Classes (WFC) y muchos de los paquetes com.ms.\*; vea [Bibliotecas de clases compatibles](#) para obtener más información.

Sin embargo, Visual J#:

- No compila código fuente de Java como código de bytes de Java (es decir, archivos .class).
- No admite el desarrollo de subprogramas ni la capacidad de alojar subprogramas en exploradores ni de crear aplicaciones que se ejecuten en Java Virtual Machine.
- No admite JNI (interfaz nativa de Java), RNI (interfaz nativa sin formato) ni RMI (invocación de métodos remotos).

Microsoft Visual J# no se utiliza para el desarrollo de aplicaciones que se ejecuten en Java Virtual Machine. Las aplicaciones y los servicios generados con Visual J# se ejecutarán solamente en .NET Framework. Microsoft ha desarrollado Visual J# de manera independiente. No está refrendado ni aprobado por Sun Microsystems, Inc.

## Vea también

[Introducción a Visual J#](#) | [Arquitectura de Visual J#](#)

# Características de Visual J# Standard

En este tema se resumen y describen las características incluidas en Visual J# .NET Standard.

## Requisitos previos para la plataforma

Cuando instale Visual J# .NET Standard, la secuencia de instalación detectará automáticamente la configuración del sistema y le pedirá que instale los componentes básicos del producto.

## Componentes básicos

Dependiendo de la configuración del sistema, se instalan de forma automática uno o varios de los siguientes componentes básicos:

- Windows 2000 Service Pack 3
- Microsoft Windows Installer 2.0
- Cliente de Extensiones Web de Microsoft FrontPage 2000
- Archivos en tiempo de ejecución para la instalación
- Microsoft Internet Explorer 6
- Microsoft Data Access Components 2.7
- Microsoft .NET Framework 1.1
- Microsoft Visual J# .NET Redistributable Package 1.1

**Nota** El paquete Microsoft Visual J# .NET Redistributable Package se usa en las aplicaciones y servicios desarrollados con Visual J#. Para obtener más información, vea [Información para los desarrolladores](#). Para obtener más información sobre la implementación, vea [Implementar aplicaciones de Visual J#](#).

## Common Language Runtime y .NET Framework

.NET Framework es un entorno multilenguaje para generar, implementar y ejecutar servicios Web XML y aplicaciones Web. Consta de tres partes principales:

- Common Language Runtime
- Clases de programación unificadas
- ASP.NET

## Lenguajes y bibliotecas

Están disponibles los siguientes recursos de lenguajes y bibliotecas.

- Compilador de Visual J#

Utiliza VJC.EXE para compilar archivos de código fuente de J# desde el Entorno de desarrollo integrado (IDE) o desde la línea de comandos.

El compilador proporciona compatibilidad con las extensiones de Microsoft® Visual J++® 6.0, incluidos los delegados, J/Dirct®, compatibilidad con el atributo **@com** para la interoperabilidad de Java y COM, compilación condicional y muchos otros atributos, como **@hidden** y **@security**.

- Bibliotecas de Visual J#

Ofrece una funcionalidad equivalente a la de la mayoría de las bibliotecas de JDK 1.1.4, las bibliotecas WFC (Windows Foundation Classes) y las clases especificadas en el currículo de becas avanzadas de informática (Advanced Placement Curriculum for Computer Science)

- Compilador de C# y tiempo de ejecución

Compila archivos de código fuente de C# desde la línea de comandos mediante CSC.EXE.

- Compilador de C++

Compila archivos de código fuente de C++ desde la línea de comandos mediante CL.EXE.

- Compilador y motor de tiempo de ejecución de Visual Basic

Compila archivos de código fuente de Visual Basic desde la línea de comandos mediante VBC.EXE.

## Entornos de proyecto y de edición

Están disponibles los siguientes entornos de proyectos y de edición.

- [Sistema de proyectos administrados de Visual J#](#)

El compilador traduce el código fuente a lenguaje intermedio de Microsoft (MSIL), un conjunto de instrucciones independientes de la CPU que puede convertirse en código nativo de forma eficaz.

- Complementos del entorno de desarrollo integrado (IDE) de Visual Studio

Amplían el IDE de Visual Studio agregando funcionalidad que no se encuentra en el producto básico por medio de complementos COM. Para obtener más información, vea [Crear complementos y asistentes](#).

## Depuración

En los temas siguientes se proporciona información sobre depuración para entornos de programación específicos.

- [Depurar código administrado](#)

Proporciona información general sobre la depuración de aplicaciones administradas.

- [Depurar los tipos de proyecto de Visual Studio](#)

Explica cómo depurar aplicaciones para Windows, bibliotecas de clases, la biblioteca de controles de Windows, aplicaciones Web, servicios Web XML, aplicaciones de consola y servicios de Windows.

- [Depurar en Visual J#](#)

Trata cuestiones específicas de la depuración en Visual J#.

- [Depurador en tiempo de ejecución](#)

Ofrece documentación sobre el Depurador de .NET Framework SDK, que ayudará a los proveedores de herramientas y a los desarrolladores de herramientas a encontrar y corregir errores de programas para Common Language Runtime de .NET Framework.

Para obtener más información, vea [Configuraciones de versión de depuración y versión de lanzamiento](#).

## Implementación de aplicaciones

Las aplicaciones y controles creados para .NET Framework necesitan que .NET Framework esté instalado en el equipo en el que se ejecute la aplicación o control.

- Proyecto de instalación

Los proyectos de instalación sirven para crear archivos de Windows Installer (.msi), que se utilizan con el fin de distribuir la aplicación para su instalación en otro equipo o servidor Web.

## Asistentes para aplicaciones

La mejor manera de crear proyectos nuevos es utilizar los asistentes para aplicaciones de Visual J#. Los asistentes para aplicaciones de Visual J# trabajan conjuntamente con los marcos de trabajo de aplicación y las bibliotecas para crear programas iniciales modelados a partir de plantillas de proyecto.

Asistente para aplicaciones	Descripción
Aplicación para Windows	Crea una aplicación de Visual J# con una interfaz de usuario de Windows.
Biblioteca de clases	Crea un proyecto para crear clases que se pueden utilizar en otras aplicaciones.
Aplicación de consola	Crea una aplicación de Visual J# con una interfaz de línea de comandos.
Aplicación Web ASP.NET	Crea una aplicación de Visual J# con una interfaz de usuario Web.
Servicio Web ASP.NET	Crea un servicio Web XML con Visual J#, al que podrán tener acceso otras aplicaciones.

Aplicación Web ASP.NET Mobile	Crea una aplicación que se puede ver en PDA, teléfonos móviles y otros dispositivos móviles.
Nuevo proyecto en carpeta existente	Crea un proyecto vacío en una carpeta existente.
Proyecto vacío	Crea un proyecto vacío para crear una aplicación local.
Proyecto Web vacío	Crea un proyecto vacío para crear una aplicación Web.

Para obtener más información, vea [Proyectos de Visual Basic y Visual C#](#).

## Otras herramientas

En la edición Standard se incluyen el asistente, la herramienta y los diseñadores siguientes:

- Asistente para actualizar

Ayuda en la actualización de proyectos de Visual J++ a Visual J#. El asistente se inicia al abrir una solución o un proyecto de Visual J++ en el entorno de Visual J#.

- Conversor binario de Visual J# (Jblmp.exe)

Convierte determinados archivos de código de bytes de Java (.class) en Lenguaje intermedio de Microsoft® (MSIL).

Permite a los desarrolladores convertir en ensamblados de MSIL la mayoría de las bibliotecas y aplicaciones del JDK 1.1.4 disponibles únicamente como archivos de código de bytes, así como ejecutarlas en Visual J#.

- Diseñador de Web Forms

Proporciona una solución de desarrollo rápido para crear aplicaciones Web. Los formularios Web Forms son una tecnología ASP.NET que se utiliza para crear páginas Web programables.

- Diseñador de Windows Forms

Proporciona una solución de desarrollo rápido para crear aplicaciones para Windows. Los formularios Windows Forms son la nueva plataforma para el desarrollo de aplicaciones para Microsoft Windows basadas en .NET Framework.

- Diseñador XML

Proporciona un conjunto de herramientas visuales para trabajar con esquemas XML, conjuntos de datos ADO.NET y documentos XML.

## Ayuda en pantalla y documentación

La Colección de ayuda combinada de Visual Studio .NET (VSCC) contiene la documentación estándar de Visual Studio y también documentación de paquetes o complementos de terceros instalados.

## Tecnologías de servidor

Microsoft Data Engine (MSDE) es un motor de datos totalmente compatible con SQL Server que se utiliza para generar soluciones de escritorio y soluciones compartidas, y que proporciona la vía de migración más sencilla a SQL Server 7.0. Para obtener más información, vea <http://msdn.microsoft.com/vstudio/msde/genfaq.asp>.

## Kits de desarrollo de software (SDK)

El SDK de .NET Framework ofrece muchos recursos, incluidos archivos DLL, herramientas y ejemplos, que permiten a los programadores generar aplicaciones y servicios basados en Web eficaces y escalables que aprovechan la nueva tecnología .NET Framework. Para obtener más información consulte [.NET Framework SDK](#).

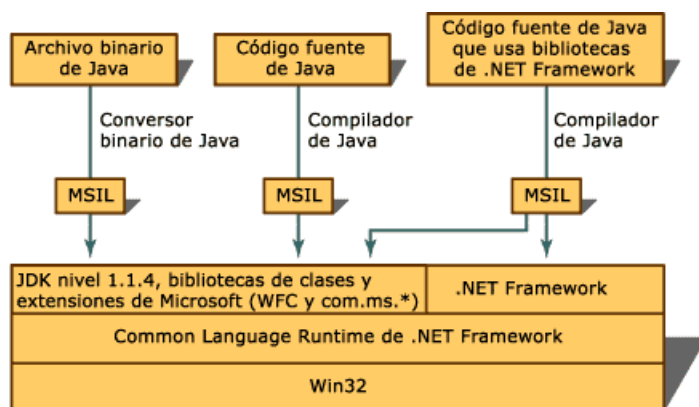
## Vea también

[Introducción a Visual J#](#) | [Ediciones de Visual Studio .NET](#) | [Características de Visual Basic Standard](#) | [Características de Visual C# Standard](#) | [Características de Visual C++ Standard](#)

# Arquitectura de Visual J#

Visual J# incluye:

- Compilador de Visual J# (vjc.exe)
- Conversor binario de Visual J# (Jblmp.exe)
- Un conjunto de bibliotecas de clases desarrolladas de manera independiente y diseñadas para proporcionar la funcionalidad de la mayoría de las bibliotecas de clases de JDK nivel 1.1.4 y muchas de las clases del paquete **java.util** de JDK 1.2 especificadas en el currículo de becas avanzadas de informática (Advanced Placement curriculum for Computer Science) del sistema College Board en Estados Unidos.
- Compatibilidad con las extensiones de Microsoft® de Visual J++ 6.0®, incluido Windows® Foundation Classes (WFC) y muchos de los paquetes com.ms.\*.
- Las bibliotecas de clases que se distribuyen con Visual J# son capas basadas en .NET Framework y Common Language Runtime, como se muestra en la siguiente ilustración.



Los archivos de código fuente de Java desarrollados con Visual J++ 6.0 se pueden compilar como un archivo ejecutable administrado .NET con el compilador de Visual J#. Determinadas aplicaciones disponibles sólo con formato de código de bytes se pueden convertir estáticamente en un archivo ejecutable administrado .NET con la herramienta Conversor binario de Visual J# .NET. Se pueden desarrollar aplicaciones de Visual J# nuevas para utilizar bibliotecas de clases de JDK nivel 1.1.4 compatibles, extensiones de Microsoft en Visual J++ 6.0 (por ejemplo, WFC y com.ms.\*), así como .NET Framework.

## Vea también

[Introducción a Visual J#](#) | [Información general sobre Visual J#](#)

## Buscar información de referencia sobre Java y JDK

La documentación de Visual J# no incluye material de referencia para las bibliotecas de clases de Java o JDK nivel 1.1.4.

Para obtener más información sobre las bibliotecas de clases de Java y JDK nivel 1.1.4, vea MSDN Online Library en <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vjcore98/html/vjovrdocumentationmap.asp>.

### **Vea también**

[Introducción a Visual J#](#)



# Buscar material de referencia en la documentación de Visual Studio .NET

Gran parte de la documentación de Visual Studio es aplicable a Visual J#. La documentación de Visual J# señala las diferencias o excepciones donde procede. Para buscar temas de referencia útiles, vea los siguientes temas:

- [Material de referencia en la documentación de Visual Studio .NET](#)
- [Material de referencia en la documentación de Visual Basic y Visual C#](#)
- [Material de referencia en la documentación de .NET Framework](#)

## Vea también

[Visual J#](#) | [Introducción a Visual J#](#)

# Material de referencia en la documentación de Visual Studio .NET

Las siguientes son algunas referencias útiles de la documentación de Visual Studio .NET:

## [Lo nuevo en Visual Studio .NET](#)

Descubra las nuevas características de Visual Studio .NET, incluidas las características del entorno de desarrollo integrado (IDE), programas de ejemplo y documentación del producto.

## [Obtener asistencia](#)

Obtenga más información acerca de las características de accesibilidad, la ubicación de archivos Léame, archivos de asistencia y consejos para utilizar la documentación en pantalla.

## [Desarrollar con Visual Studio .NET](#)

Obtenga más información acerca del diseño, desarrollo, depuración, prueba, implementación y administración de aplicaciones creadas mediante Visual Studio.

## [Diseñar aplicaciones distribuidas](#)

Información sobre decisiones de diseño de aplicaciones, como la arquitectura del sistema, el diseño de la base de datos y consideraciones internacionales.

## **Vea también**

[Buscar material de referencia en la documentación de Visual Studio .NET](#)

# Material de referencia en la documentación de Visual Basic y Visual C#

Muchas de las características de Visual J# son similares a las de Visual Basic y Visual C#. Por tanto, la mayor parte de la documentación de Visual Basic y Visual C# es aplicable a Visual J#. Las siguientes son algunas referencias útiles de la documentación de Visual Basic y Visual C#:

## [Lo nuevo en Visual Basic y Visual C#](#)

Trata las nuevas características de diferentes áreas, incluidos el desarrollo Web, el acceso a datos y la creación de componentes.

## [Gráfico de decisiones](#)

Proporciona una guía interactiva que le ayuda a elegir el enfoque o la tecnología adecuados, así como vínculos a más información.

## [Tareas comunes en Visual Basic y Visual C#](#)

Proporciona vínculos a temas sobre tareas de programación comunes, incluidos proyectos de administración, creación de servicios Web XML, creación de componentes, trabajo con XML y datos, depuración, implementación de aplicaciones y mucho más.

## [Administrar proyectos](#)

Describe el sistema de proyectos para la administración de aplicaciones a medida que se escriben.

## [Crear aplicaciones](#)

Describe cómo crear y configurar aplicaciones Windows, aplicaciones Web y servicios Web XML, y le ayuda a personalizar estas aplicaciones para un público internacional.

## [Acceso a datos](#)

Trata aspectos de la incorporación de funcionalidad de acceso a datos a las aplicaciones, utilizando la nueva tecnología de generación de acceso a datos: ADO.NET.

## [Programar con componentes](#)

Explica cómo crear todos los tipos de componente, incluidos los utilizados en servidores.

## **Vea también**

[Buscar material de referencia en la documentación de Visual Studio .NET](#)

# Material de referencia en la documentación de .NET Framework

Las siguientes son algunas referencias útiles de la documentación de .NET Framework:

## [Introducción](#)

Familiariza al usuario con los fundamentos de .NET Framework, a la vez que ofrece estrategias que ayudan a utilizar .NET Framework SDK de forma rápida y eficaz.

## [Dentro de .NET Framework](#)

Describe conceptos clave de .NET Framework, como Common Language Runtime, el sistema de tipos comunes (CTS), la interoperabilidad entre lenguajes, la ejecución administrada, los ensamblados y la seguridad.

## [Programar con .NET Framework](#)

Explica tareas de programación comunes que se aplican a varias aplicaciones de .Net Framework. En los temas se explica cómo obtener acceso a datos, extender metadatos, controlar e iniciar excepciones, procesar transacciones y proteger aplicaciones.

## [Crear aplicaciones](#)

Proporciona introducciones instructivas y procedimientos paso a paso detallados para crear determinadas categorías de aplicaciones, como aplicaciones ASP.NET y aplicaciones de formularios Windows Forms. En esta sección también se describe la arquitectura en tiempo de diseño de .NET Framework, a la vez que se muestra cómo se puede agregar funcionalidad en tiempo de diseño a las aplicaciones.

## [Depurar y generar perfiles de aplicaciones](#)

Explica cómo probar, optimizar y generar perfiles de aplicaciones .NET Framework y el entorno de la aplicación. En esta sección se incluye información tanto para administradores como para programadores.

## [Implementar aplicaciones](#)

Muestra cómo crear una aplicación autodescriptiva y autosuficiente, ya se trate de una aplicación de formularios Windows Forms, una aplicación ASP.NET o un control para descargar.

## [Configurar aplicaciones](#)

Explica cómo los programadores y los administradores pueden aplicar valores a varios tipos de archivos de configuración.

## **Vea también**

[Buscar material de referencia en la documentación de Visual Studio .NET](#)

# Información para los desarrolladores

Microsoft Visual J# es una herramienta de desarrollo que pueden utilizar los desarrolladores que estén familiarizados con la sintaxis de Java para generar aplicaciones y servicios en .NET Framework. Integra la sintaxis de Java en el shell de Visual Studio .NET. Visual J# admite también la funcionalidad de Visual J++ 6.0, incluidas las extensiones de Microsoft.

Visual J# no se utiliza para el desarrollo de aplicaciones que se ejecuten en Java Virtual Machine. Las aplicaciones y los servicios generados con Visual J# se ejecutarán solamente en .NET Framework.

Microsoft Visual J# .NET Redistributable Package es el paquete redistribuible para Microsoft Visual J#. Este paquete ejecuta únicamente aplicaciones y servicios desarrollados con Visual J#; las aplicaciones de Java escritas con otras herramientas de desarrollo de Java no se ejecutarán con Microsoft Visual J# .NET Redistributable Package.

Microsoft ha desarrollado Visual J# y Microsoft Visual J# .NET Redistributable Package de manera independiente y no están refrendados ni aprobados por Sun Microsystems, Inc.

# Tutoriales de Visual J#

Los siguientes tutoriales pretenden familiarizarle con el uso de las características de Visual Studio .NET para escribir aplicaciones con Visual J#.

## Tutorial sobre aplicaciones distribuidas

### [Crear una aplicación distribuida con Visual J#](#)

Muestra cómo crear una aplicación que conste de tres niveles lógicos: datos, objeto comercial e interfaz de usuario.

## Tutoriales sobre datos

### [Crear un conjunto de datos con tablas, claves y relaciones uno a varios con Visual J#](#)

Describe los pasos necesarios para crear un conjunto de datos y utilizarlo para validar datos en una relación de uno a varios o en una relación de claves.

### [Controlar una excepción de concurrencia con Visual J#](#)

Describe cómo utilizar el objeto **DBConcurrencyException** para identificar las excepciones de concurrencia y el registro real que provocó el error.

### [Asignar tablas del origen de datos a tablas de conjunto de datos con Visual J#](#)

Describe cómo cargar datos de una base de datos en un conjunto de datos basándose en un esquema distinto utilizando una tabla de adaptador de datos y asignaciones de columnas.

### [Crear un archivo XML con un esquema XML asociado en Visual J#](#)

Describe cómo crear un archivo XML y un esquema, asociar ambos y trabajar con el archivo XML en la vista XML del diseñador.

### [Crear un esquema XML con el Diseñador XML en Visual J#](#)

Explica cómo se crea un esquema de pedido XML como parte de una aplicación para Windows.

### [Leer datos XML en un conjunto de datos con Visual J#](#)

Muestra cómo cargar datos XML en un conjunto de datos, mostrarlos en un control **DataGrid** de un formulario Windows Forms y, a continuación, mostrar un esquema XML del archivo XML en un cuadro de texto.

## Tutoriales sobre los formularios Windows Forms

### [Crear una aplicación para Windows con Visual J#](#)

Explica el proceso para escribir y ejecutar un formulario Windows Forms sencillo.

### [Crear un formulario Windows Forms Principal-Detalle con Visual J#](#)

Describe cómo crear una relación entre primario y secundario en un conjunto de datos y mostrar registros relacionados en un formulario Windows Forms.

### [Crear menús contextuales dinámicos en formularios Windows Forms con Visual J#](#)

Explica detalladamente, mediante un ejemplo, cómo crear un único menú contextual para dar servicio a dos o más controles diferentes.

### [Cambiar de estructura de menús en formularios Windows Forms basándose en el estado de una aplicación con Visual J#](#)

Ofrece indicaciones para cambiar entre objetos MainMenu mediante programación.

### [Mostrar la herencia de Visual con Visual J#](#)

Describe cómo se crea un formulario Windows Forms base y cómo se compila en una biblioteca de clases. Se importa la biblioteca de clases a otro proyecto y se crea un nuevo formulario que hereda del formulario base.

### [Crear una interfaz de usuario de varios paneles con formularios Windows Forms con Visual J#](#)

Explica cómo se crea una interfaz de usuario de varios paneles similar a la que se utiliza en Microsoft Outlook.

### [Acceso a datos sencillo en un formulario Windows Forms con Visual J#](#)

Describe los pasos básicos requeridos para tener acceso a datos de SQL Server a través de un conjunto de datos en un formulario Windows Forms de lectura y escritura.

### [Mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

Explica cómo crear un conjunto de datos que contenga registros seleccionados, basándose en criterios que proporcionan los usuarios en tiempo de ejecución en un formulario Windows Forms.

### [Llamar a los servicios Web XML desde los formularios Windows Forms con Visual J#](#)

Explica cómo llamar a métodos de servicios Web desde una aplicación para Windows.

### [Crear una aplicación de servicios de Windows en el Diseñador de componentes con Visual J#](#)

Describe cómo se crea un archivo ejecutable de ejecución larga que carece de interfaz de usuario.

## Tutoriales sobre los formularios Web Forms

### [Crear una página básica de formularios Web Forms con Visual J#](#)

Muestra las técnicas básicas para crear una página de formularios Web Forms, agregarle controles y ejecutarla.

### Crear una aplicación Web utilizando Visual J#

Explica cómo se escribe una aplicación de formularios Web Forms, para después integrar un componente de objeto comercial que aplique un descuento del 10% en compras.

### Crear una aplicación Web utilizando un objeto comercial de otro fabricante con Visual J#

Explica cómo se escribe una aplicación de formularios Web Forms utilizando extensiones administradas de C++, para después integrar un componente de objeto comercial que aplique un descuento del 10 % en compras.

### Validar los datos proporcionados por el usuario en una página de formularios Web Forms con Visual J#

Explica cómo utilizar controles de validación de formularios Web Forms para comprobar entradas de usuario sin código (por ejemplo, entradas requeridas, tipos de datos, patrones y valores específicos).

### Mostrar datos en una página de formularios Web Forms con Visual J#

Muestra un escenario de acceso a datos básico, con instrucciones paso a paso para crear una página de formularios Web Forms que muestre datos en un control de cuadrícula.

### Crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J#

Describe cómo usar un comando de datos y un lector de datos de ADO.NET para proporcionar acceso optimizado a datos de sólo lectura en una página de formularios Web Forms.

### Actualizar datos mediante una consulta de actualización de bases de datos en los formularios Web Forms con Visual J#

Describe las técnicas básicas para crear una página de formularios Web Forms que use un comando de datos para actualizar la base de datos.

### Utilizar un control Web DataGrid para leer y escribir datos con Visual J#

Proporciona instrucciones paso a paso sobre cómo utilizar un control de cuadrícula para permitir que los usuarios vean y editen datos.

### Crear acceso a datos paginado mediante una página de formularios Web Forms con Visual J#

Explica cómo crear una página de formularios Web Forms que permita a los usuarios ver registros de datos (unos pocos registros cada vez) avanzando y retrocediendo para ver los registros posteriores y anteriores.

### Crear un control Web de usuario con Visual J#

Explica cómo crear un control de usuario de formularios Web Forms (es decir, una página configurada de tal modo que actúe como un control) y utilizarlo en otra página.

### Convertir una página de formularios Web Forms en un control de usuario con Visual J#

Explica cómo se convierte una página básica de formularios Web Forms en un control de usuario que puede alojarse en otra página.

### Crear un control Web personalizado con Visual J#

Explica cómo crear un control personalizado, agregarlo al cuadro de herramientas y utilizarlo en una página de formularios Web Forms.

### Mostrar un documento XML en una página de formularios Web Forms mediante transformaciones con Visual J#

Proporciona información detallada sobre cómo leer un archivo XML y mostrarlo de diferentes maneras en una página de formularios Web Forms.

## Crear y obtener acceso a los tutoriales de los servicios Web XML

### Crear un servicio Web XML con Visual J#

Explica el proceso para escribir un servicio Web XML sencillo utilizando la plantilla de proyecto de servicio Web de ASP.NET.

### Acceso a un servicio Web XML con un cliente de formularios Web Forms de Visual J#

Explica el proceso para escribir un cliente de formularios Web Forms para un servicio Web XML.

### Acceso a un servicio Web XML con un cliente de Windows de Visual J#

Explica el proceso para escribir un cliente de formularios Windows Forms para un servicio Web XML.

## Tutoriales sobre la creación de componentes y controles

### Crear un componente con Visual J#

Muestra cómo programar en J# un componente sencillo. Ilustra la interacción entre cliente y componente, el período de duración de un objeto y las referencias circulares, la depuración de clientes y componentes, y el uso de métodos compartidos y métodos de instancia.

### Crear un componente sencillo multiproceso con Visual J#

Muestra la creación de un componente multiproceso, explicando cómo funcionan los subprocesos y cómo coordinar varios subprocesos en un componente.

### Crear un control de usuario con Visual J#

Muestra el desarrollo de un sencillo control de usuario que se hereda de la clase **UserControl** y cómo heredar de un control de usuario establecido.

### Crear una clase de colección propia con Visual J#

Proporciona instrucciones detalladas para implementar su propia clase de colección con establecimiento inflexible de tipos.

### Heredar de un control de Windows Forms con Visual J#

Muestra cómo crear un sencillo control de botón heredado. Este botón heredará la funcionalidad del botón de Windows Forms

estándar y expondrá un miembro personalizado.

## Tutoriales sobre el componente de servicios Framework

### [Explorar registros de eventos, orígenes de eventos y entradas con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar registros de eventos y sus entradas, y escribir entradas adicionales en ellos.

### [Instalar un componente de registro de eventos con Visual J#](#)

Explica cómo se utiliza un componente de instalación para configurar recursos de servidor necesarios para la instancia del componente **EventLog**.

### [Crear una cola y trabajar con mensajes en Visual J#](#)

Explica cómo utilizar una instancia de componente **MessageQueue** para que interactúe con colas de mensajes de Windows.

### [Cambiar y recuperar valores de los contadores de rendimiento en Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar valores sin formato y valores calculados de un contador mediante el componente **PerformanceCounter**.

### [Recuperar categorías y contadores con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar listas de categorías de contadores de rendimiento, así como sobre los contadores que éstas contienen.

### [Administrar un proceso de Windows con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo crear una instancia de un componente **Process** y utilizarlo para interactuar con un proceso de Windows.

### [Reaccionar a eventos del sistema de archivos con Visual J#](#)

Explica cómo utilizar una instancia de componente **FileSystemWatcher** para supervisar archivos y directorios, y reaccionar cuando se produzcan cambios.

### [Agregar objetos Active Directory con Visual J#](#)

Explica cómo utilizar el componente **DirectorySearcher** para interactuar con las estructuras del sistema de archivos de Active Directory.

## Tutorial sobre propiedades dinámicas

### [Almacenar y recuperar propiedades dinámicas con Visual J#](#)

Describe los principales pasos que hay que seguir para almacenar y recuperar valores de propiedades.

## Tutoriales sobre localización

### [Localización de formularios Windows Forms con Visual J#](#)

Proporciona instrucciones detalladas de ejemplo sobre cómo adaptar formularios Windows Forms.

### [Localizar páginas de formularios Web Forms con Visual J#](#)

Proporciona instrucciones detalladas de ejemplo sobre cómo adaptar páginas de formularios Windows Forms.

## Tutoriales sobre accesibilidad

### [Crear una aplicación para Windows accesible con Visual J#](#)

Explica cómo crear una aplicación para Windows optimizada para distintos aspectos de accesibilidad.

### [Crear una aplicación Web accesible con Visual J#](#)

Explica cómo crear páginas Web accesibles para personas con discapacidades y usuarios con conexiones lentas o exploradores de texto únicamente.

## Información adicional

### [Visual J# .NET](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

### [Ejemplos de Visual J#](#)

Código fuente de ejemplo que muestra cómo escribir aplicaciones para .NET Framework utilizando Visual J#.

### [Tutoriales de Visual Studio](#)

Guías paso a paso para crear diferentes tipos de aplicaciones y componentes utilizando los distintos lenguajes de programación de Visual Studio .NET.



# Tutorial sobre aplicaciones distribuidas

El tutorial de esta sección explica cómo crear una aplicación distribuida de niveles múltiples, que tenga una aplicación de cliente Windows enriquecida y una interfaz de exploración.

## En esta sección

### [Crear una aplicación distribuida con Visual J#](#)

Muestra cómo crear una aplicación que conste de tres niveles lógicos: datos, objeto comercial e interfaz de usuario. El nivel datos es una base de datos en SQL Server. El nivel de objeto comercial controla el acceso a los datos y su distribución entre los clientes. El nivel de interfaz de usuario consta de una aplicación basada en Web y una aplicación para Windows tradicional.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, trabajar con formularios Windows Forms y Web Forms, y realizar tareas relacionadas con datos.

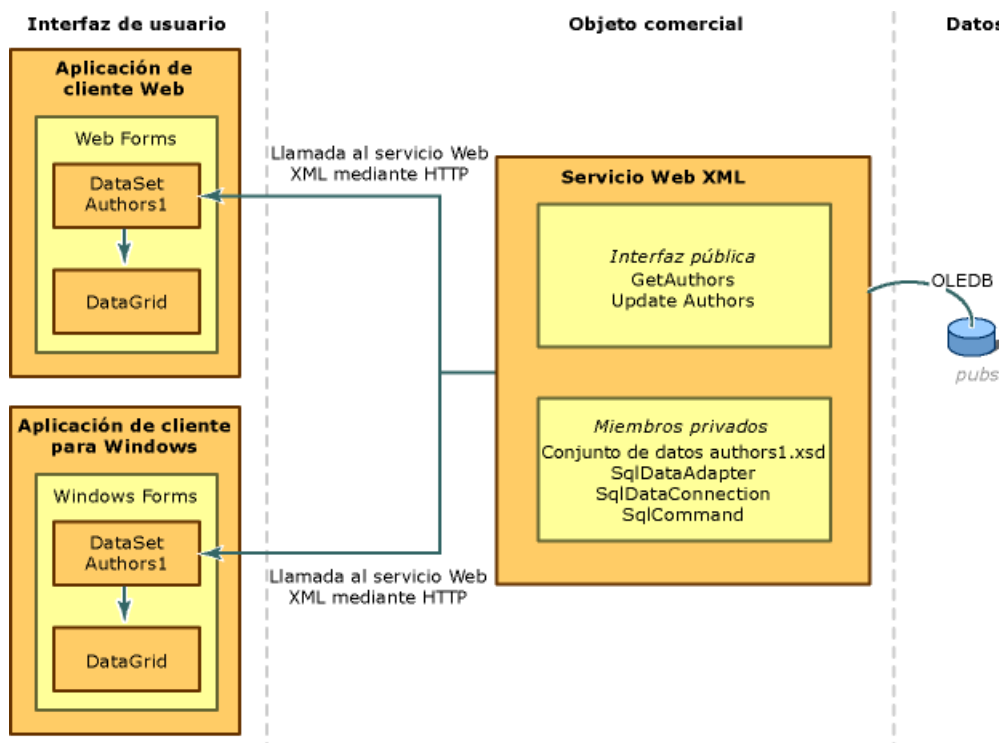
### [Diseñar aplicaciones distribuidas](#)

Información sobre decisiones de diseño de aplicaciones, como la arquitectura del sistema, el diseño de la base de datos y consideraciones internacionales.

# Tutorial: crear una aplicación distribuida con Visual J#

En este tutorial, se creará una aplicación distribuida multinivel. La aplicación constará de tres niveles lógicos: datos, objeto comercial e interfaz de usuario. El nivel datos es una base de datos en SQL Server. El nivel de objeto comercial manejará el acceso a los datos y su distribución entre los clientes. El nivel de interfaz de usuario constará de una aplicación basada en Web y una aplicación de Windows tradicional. La siguiente ilustración describe la arquitectura de la aplicación.

## Arquitectura de la aplicación distribuida que se crea



La aplicación que va a generar es una aplicación de datos sencilla con búsqueda y edición. Generará un cliente basado en Windows y en Web para mostrar la tabla Authors desde la base de datos de ejemplo Pubs de SQL Server. Para la parte Web se utilizará el Diseñador de Web Forms para crear una página Web que es compatible con un explorador estándar HTML 3.2. En el servidor, el código de formularios Web Forms llamará a un servicio Web XML para recuperar datos que contienen información de la tabla Authors de la base de datos. Para la parte de Windows, se generará una aplicación para Windows que comunicará con este mismo servicio Web XML para recuperar un conjunto de datos que contiene información del autor. La comunicación con el servicio Web XML se administra mediante HTTP y XML.

## Requisitos del sistema

Para poder completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Pubs de SQL Server, configurada para la autenticación de Windows integrada. La base de datos Pubs es una de las bases de datos de ejemplo que se puede instalar con SQL Server.
- Conocimientos básicos sobre cómo manejar datos en Visual Studio .NET. Para obtener más información, vea [Introducción al acceso a datos con ADO.NET](#).

## Procesos para crear una aplicación distribuida

Un posible escenario para desarrollar una aplicación distribuida es crear un nivel cada vez; por ejemplo, se empieza con el nivel de datos, después se pasa al objeto comercial de nivel medio y, por último, se crea la interfaz de usuario. Para este tutorial, los datos ya se han generado y están disponibles en la base de datos Pubs de SQL Server. Por lo tanto, el tutorial comienza con la creación del objeto comercial —el servicio Web XML— y, a continuación, se generan las dos interfaces de usuario —una página de formularios Web Forms y un formulario Windows Forms.

El proceso de este tutorial es el siguiente:

1. [Crear el objeto comercial de nivel medio](#)
  - a. [Crear un proyecto Servicio Web ASP.NET](#)
  - b. [Crear y configurar una conexión de base de datos y un esquema de conjunto de datos](#)

- c. [Exponer el conjunto de datos desde el objeto comercial](#)
2. [Crear la interfaz de usuario](#)
  - a. [Crear una interfaz de usuario de Windows](#)
  - b. [Crear una interfaz de usuario Web](#)
3. [Implementar la solución](#)

## Crear el objeto comercial de nivel medio

El objeto comercial que se crea se ejecutará en un servidor Web para suministrar el rendimiento y la escalabilidad necesarios para una aplicación distribuida. Además, se implementará el objeto comercial como un servicio Web XML para que así los clientes puedan utilizar protocolos estándar de Internet para comunicarse con el objeto comercial desde cualquier plataforma. Para obtener información, vea [Programar el Web con servicios Web XML](#).

En este tutorial, el componente de servicio Web XML guarda la conexión de datos y la definición de conjunto de datos. A continuación, los métodos del servicio Web XML se agregarán para exponer el conjunto de datos, lo que permite a otras aplicaciones ver y modificar el conjunto de datos.

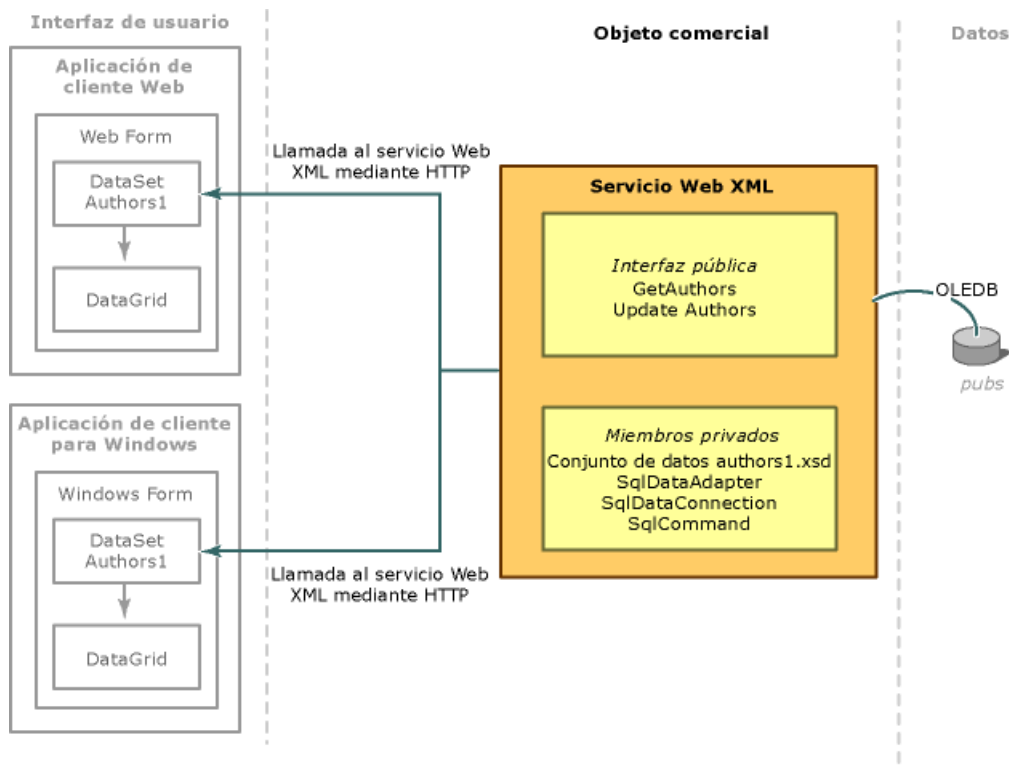
La plantilla del proyecto Servicio Web ASP.NET es un proyecto basado en Web diseñado para crear componentes de nivel medio que muestran sus interfaces como servicios Web XML. Para obtener más información, vea [Proyectos Servicio Web ASP.NET en Visual Studio](#).

El servicio Web XML mostrará dos métodos. El primero, **GetAuthors**, devolverá un conjunto de datos desde la base de datos. El segundo, **UpdateAuthors**, actualizará la base de datos con cambios procedentes del usuario. Varios miembros privados se crean para implementar estos métodos. Entre ellos se incluyen:

- Una instancia de un conjunto de datos
- Un adaptador de datos
- Una conexión de datos
- Varios objetos Command para recuperar datos de la base de datos y enviarle actualizaciones

El siguiente diagrama describe el servicio Web XML.

### El nivel intermedio en la aplicación distribuida



### Para crear un proyecto Servicio Web ASP.NET

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Servicio Web ASP.NET** en el panel **Plantillas**.
3. En el cuadro **Ubicación**, escriba el nombre del servidor Web con el nombre del proyecto,

<http://NombreDeServidor/AuthorsWebService> y, después, haga clic en **Aceptar**.

**Sugerencia** Si el servidor Web está en su equipo, puede utilizar el nombre de servidor **LOCALHOST**.

**Sugerencia** Si existen problemas al crear el proyecto Servicio Web ASP.NET, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

4. El proyecto **AuthorsWebService** se agrega a la solución. El Diseñador de componentes para **Service1.aspx** aparece en el entorno de desarrollo.
5. En el Explorador de soluciones, haga doble clic en **Service1.aspx** para seleccionarlo.
6. En la ventana Propiedades, asigne a la propiedad **Name** de **Service1** el valor **AuthorsService**.
7. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el archivo **Service1.aspx**, seleccione **Cambiar nombre**, y cambie el nombre del archivo **AuthorsService.aspx**, para que coincida con el nombre del servicio.

En este componente se creará una conexión para guardar datos y obtener una instancia de los datos mediante un conjunto de datos.

### Crear y configurar una conexión de base de datos y un esquema de conjunto de datos

Se agregan dos objetos al servicio Web XML: un objeto [SqlDataAdapter](#) y un objeto [SqlConnection](#). El objeto de conexión crea una conexión nueva a la base de datos mientras el adaptador de datos consulta la base de datos y alimenta un objeto [DataSet](#) con los resultados. El adaptador de datos también actualiza la base de datos con los cambios realizados por el usuario en el conjunto de datos. Al utilizar el adaptador de datos, se creará un conjunto de datos para guardar una instancia de la información que se presenta en la base de datos. El conjunto de datos se utilizará más adelante en el tutorial para mostrar datos en los formularios Windows Forms y en la página de formularios Web Forms. Para obtener información general, vea [Introducción a las aplicaciones distribuidas y a la integración de datos](#).

### Para crear una conexión de base de datos y un adaptador de datos

1. Haga clic en el Explorador de servidores.

**Nota** De forma predeterminada, el Explorador de servidores aparece como una ficha vertical en la parte izquierda del entorno de programación. Si la ficha no está visible, haga clic en el menú **Ver** y elija **Explorador de servidores**.

2. En el Explorador de servidores, haga clic con el botón secundario del *mouse* (ratón) en el nodo **Conexiones de datos** y elija **Agregar conexión** en el menú contextual.
3. En la ficha **Conexión** del cuadro de diálogo **Propiedades de vínculo de datos**, introduzca el nombre de SQL Server donde está instalada la base de datos pubs. Si posee SQL Server en el equipo local, escriba **(local)**.
4. Seleccione **Usar seguridad integrada de Windows NT** para la información de inicio de sesión.

**Nota** Si no tiene configurada la seguridad integrada en su sistema, póngase en contacto con su administrador de red.

5. Seleccione la base de datos **pubs** en la lista.
6. Haga clic en **Probar conexión** para validar la información proporcionada y después haga clic en **Aceptar** para establecer la conexión.

Aparece un nuevo nodo en el nodo de **Conexiones de datos** del Explorador de servidores.

**Nota** Si la conexión a la base de datos falla, consulte con su administrador de base de datos.

7. En el Explorador de servidores, expanda el nodo para el nuevo nodo de conexión y, a continuación, expanda el nodo **Tablas**.
8. Busque el nodo **authors** y expándalo para que muestre los campos en la tabla **authors**.
9. Mediante CTRL+Clic, seleccione los campos **au\_id**, **au\_lname**, **au\_fname** y **city**.
10. Arrastre estos campos desde el Explorador de servidores hasta la superficie de diseño. Un objeto **SqlConnection** emparejado con un objeto **SqlDataAdapter** aparece en el diseñador. Se ha creado una conexión a la base de datos, con la transferencia de información administrada por el objeto **SqlDataAdapter**. Estos componentes se configuran para mover datos asociados con la tabla **authors** dentro y fuera de la base de datos.

**Nota** Este tutorial utiliza el objeto **SqlDataAdapter**, que está optimizado para trabajar con SQL Server 7.0 o versiones posteriores. Se podría utilizar el objeto **OleDbDataAdapter** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB. Si estuviera conectado a una

base de datos que no fuera SQL Server, el sistema del proyecto crearía un objeto **OleDbDataAdapter** y un objeto **OleDbConnection**.

**Nota** En este tutorial se ha llenado el conjunto de datos con algunas de las columnas y todas las filas de la tabla **authors**. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

Es necesario configurar los valores de seguridad del proyecto para trabajar con seguridad integrada. Para ello se desactiva el acceso anónimo y se activa la suplantación. Para obtener más información, vea [Modelo de seguridad](#).

### Para configurar la autenticación de Windows integrada

Para configurar la autenticación de Windows integrada para el proyecto, es necesario cambiar los archivos del proyecto y configurar el proyecto utilizando la herramienta **Servicios de Internet Information Server**.

1. Inicie la herramienta **Servicios de Internet Information Server**. Se puede ejecutar desde **Herramientas administrativas** en el **Panel de control**. (Para obtener más información acerca de esta herramienta, vea la documentación de ayuda de Windows.)
2. Expanda el nodo de su servidor.
3. Expanda el nodo **Sitio Web predeterminado**.
4. Haga clic con el botón secundario del *mouse* (ratón) en el nodo de **AuthorsWebService** y elija **Propiedades** en el menú contextual.
5. Haga clic en la ficha **Seguridad de directorios**.
6. Haga clic en el botón **Editar** en la sección **Control de autenticación y acceso anónimo**.
7. Desactive la casilla de verificación **Acceso anónimo**.
8. Active la casilla de verificación **Autenticación de Windows integrada**. Acaba de configurar el directorio del servicio Web XML.
9. Al volver al proyecto en Visual Studio, haga doble clic en el archivo Web.config en el Explorador de soluciones.
10. Agregue la siguiente etiqueta en la línea después de la etiqueta `<system.web>` para configurar la seguridad integrada del servicio Web XML.

```
<identity impersonate="true"/>
```

### Para crear una definición de la clase de conjunto de datos

1. En el Explorador de soluciones, haga doble clic en el archivo **AuthorsService** para abrirlo en el diseñador.
2. En el menú **Datos**, elija **Generar conjunto de datos**. En el cuadro de diálogo **Generar conjunto de datos**, seleccione **Nuevo** y denomine al conjunto de datos con "authors1". No active la casilla denominada "Agregar este conjunto de datos al diseñador".

**Nota** Se crea un archivo de esquema de conjunto de datos, authors1.xsd, y se agrega al proyecto. Este archivo de esquema contiene una definición de clase para **authors1**. Esta clase, heredada de la clase **DataSet**, contiene una definición de conjunto de datos con tipo para la tabla **authors**.

3. En el menú **Archivo**, elija **Guardar todo**.

### Exponer el conjunto de datos authors1 desde el objeto comercial

El siguiente paso en este tutorial es exponer el objeto de conjunto de datos recientemente creado desde el objeto comercial. Esto pondrá a disposición el uso del conjunto de datos en aplicaciones para Windows o aplicaciones Web.

### Agregar métodos al servicio Web XML

1. En el Explorador de soluciones, haga doble clic en **AuthorsService** si no está ya abierto en el diseñador.
2. En el menú **Ver**, haga clic en **Código**.
3. Agregue un método denominado **GetAuthors** para entregar un conjunto de datos al cliente.

El método que se muestra abajo, crea un nuevo conjunto de datos **authors1** y lo rellena mediante el objeto **SqlDataAdapter** que se basa en la tabla **authors**. A continuación el método devuelve el conjunto de datos.

```
// Visual J#
/** @attribute WebMethod() */
public authors1 GetAuthors()
{
    authors1 authors = new authors1();
    sqlDataAdapter1.Fill(authors);
    sqlConnection1.Close();
    return authors;
}
```

4. Agregue un método denominado **UpdateAuthors** para propagar los cambios del cliente a la base de datos.

El método que se muestra abajo posee un parámetro del conjunto de datos **authors1** (*authorChanges*) que contiene los datos cambiados y actualizaciones de la base de datos a través del método **SqlDataAdapter.Update**. El método **Update** acepta los cambios en el conjunto de datos. Se devuelve el conjunto de datos al cliente. Entonces el cliente utilizará este conjunto de datos devueltos para actualizar su propia instancia del conjunto de datos **authors1**. Para obtener más información acerca del método **Update** y la aceptación de cambios en un conjunto de datos, vea [Introducción a los adaptadores de datos](#).

```
// Visual J#
/** @attribute WebMethod() */
public authors1 UpdateAuthors(authors1 authorChanges)
{
    if (authorChanges != null)
    {
        sqlDataAdapter1.Update(authorChanges);
        sqlConnection1.Close();
        return authorChanges;
    }
    else
    {
        return null;
    }
}
```

**Nota** En una aplicación de producción, se agregaría comprobación de errores y control de excepciones a estos métodos.

5. En el menú **Archivo**, elija **Guardar todo**.
6. En el menú **Generar**, elija **Generar solución**.
7. Implemente el servicio Web XML de manera que las referencias Web a él estén disponibles para la aplicación cliente. Para obtener más detalles, vea la sección "Implementar un servicio Web XML" en [Tutorial: crear un servicio Web XML con Visual J#](#).

En las secciones anteriores, se ha creado un objeto comercial de nivel medio que contiene un conjunto de datos enlazado a la base de datos SQL Server. Se agregó código al servicio Web XML de nivel medio **AuthorsWebService** para obtener datos desde un origen de datos y actualizar éste con los cambios. El cliente tiene acceso a estas funciones a través de los métodos del servicio Web XML **GetAuthors** y **UpdateAuthors**.

## Crear la interfaz de usuario

Después de crear un objeto comercial de nivel medio para el acceso a datos y exponerlo como un servicio Web XML, el siguiente paso es la creación de las interfaces de cliente. Existen dos escenarios en este tutorial: un formulario tradicional Windows Forms y una página de formularios Web Forms. Ambos se crearán en este ejemplo como proyectos independientes en la misma solución.

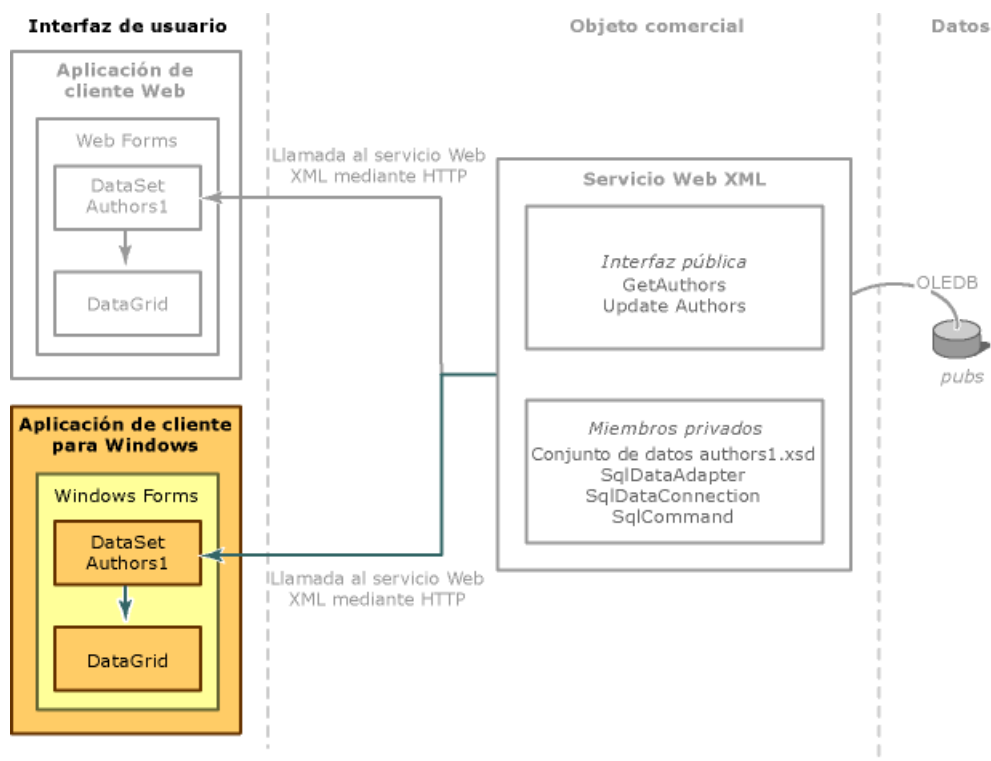
### Opción de interfaz de usuario de Windows

Una interfaz de Windows utiliza la capacidad del equipo cliente para administrar parte del procesamiento de la aplicación. Normalmente, una interfaz de Windows proporciona una mayor funcionalidad y una experiencia más rica para el usuario que una

interfaz basada en Web. Existe menos carga en el servidor que con una interfaz de usuario Web, ya que el servidor no tiene que realizar toda la lógica de la aplicación. De forma adicional, una interfaz de Windows puede beneficiarse de los recursos disponibles a través del sistema operativo, lo que incluye llamadas al sistema de archivos y al Registro.

El siguiente diagrama resalta el cliente que se implementará.

### Cliente de Windows en la aplicación distribuida



La aplicación para Windows consiste de un formulario Windows Forms que contendrá una referencia Web a **AuthorsWebService**. Los datos en la base de datos se mostrarán en un control **DataGrid** cuando se haga clic en un botón Load en el formulario. Este comportamiento de carga se implementa al llamar al método **GetAuthors** del Servicio Web XML. El control **DataGrid** permite la edición directa, con cambios en los datos que pasan de forma directa al conjunto de datos subyacente. El formulario tendrá también un botón Save. El código de este botón llamará al método **UpdateAuthors** del servicio Web XML para guardar los cambios en la base de datos.

### Para crear la aplicación para Windows

1. En el menú **Archivo**, seleccione **Agregar proyecto** y luego haga clic en **Nuevo proyecto** para abrir el cuadro de diálogo **Agregar nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows** en el panel **Plantillas**.
3. Denomine al proyecto **AuthorsWinClient** y seleccione una ubicación para el proyecto.

Se agregará el proyecto **AuthorsWinClient** a la solución. Form1 se agrega de forma automática al proyecto y aparece en el Diseñador de Windows Forms.

4. Agregue una referencia Web al proyecto Servicio Web ASP.NET que creó anteriormente:
  - a. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el proyecto **AuthorsWinClient** y después haga clic en **Agregar referencia Web** en el menú contextual.
  - b. Introduzca la ubicación del archivo .asmx del proyecto de servicio Web XML (`http://NombreDeServidor/AuthorsWebService/AuthorsService.asmx`) en el cuadro **Dirección** de la parte superior del cuadro de diálogo **Agregar referencia Web** y, a continuación, presione ENTRAR. Si tiene problemas al agregar la referencia Web, vea [Agregar referencia Web \(Cuadro de diálogo\)](#) y [Agregar y quitar referencias Web](#).
  - c. Haga clic en **Agregar referencia**.
  - d. Ahora puede crear una instancia del conjunto de datos **authors1** en la aplicación.

### Para agregar controles al formulario

1. Arrastre un control **DataGrid** desde la ficha **Windows Forms** del Cuadro de herramientas hasta el formulario.
2. Arrastre un control **Button** desde la ficha **Windows Forms** del Cuadro de herramientas hasta el formulario. Establezca la

propiedad **Name** del botón en **LoadData** y su propiedad **Text** en **Load**.

3. Arrastre otro control **Button** desde la ficha **Windows Forms** del Cuadro de herramientas hasta el formulario. Establezca la propiedad **Name** del botón en **SaveData** y su propiedad **Text** en **Save**.
4. Arrastre un objeto **DataSet** desde la ficha **Datos** del Cuadro de herramientas hasta el formulario. Aparecerá el cuadro de diálogo **Agregar conjunto de datos**. Seleccione **Conjunto de datos con tipo** y elija "AuthorsWinClient.NombreDeServidor.authors1" en la lista **Nombre**. Esta acción crea un objeto **DataSet** en la bandeja de componentes que se basa en la definición de la clase de conjunto de datos **authors1**.
5. Seleccione el control **DataSet** y establezca la propiedad **Name** en **AuthorData**.
6. Seleccione el control **DataGrid** y seleccione **AuthorData** de la lista de la propiedad **DataSource**. Seleccione **authors** de la lista de la propiedad **DataMember**. Los encabezados de columna del **DataGrid** se establecen en los nombres de columna de la tabla **authors**.

#### Para agregar código para los botones **LoadData** y **SaveData**

1. En el menú **Ver**, haga clic en **Diseñador**. Haga doble clic en el botón **LoadData** para crear un controlador de eventos vacío para el evento **Click**. Los métodos del servicio Web XML se invocan por primera vez al crear una instancia de la clase del servicio y después llamando a los métodos del servicio. En este caso, se llama al método **GetAuthors**. El conjunto de datos devuelto se combina con el conjunto de datos **AuthorData**. La propiedad **Credentials** del servicio Web XML se utiliza para pasar la identidad al servicio Web XML, que a su vez la pasa al servidor de la base de datos. Agregue al método el código que se muestra a continuación.

**Nota** Si el servicio Web XML no se ejecuta en el equipo local, necesitará reemplazar **localhost** en el ejemplo de código con el nombre del servidor que ejecuta el servicio Web XML.

```
// Visual J#
private void LoadData_Click (Object sender, System.EventArgs e)
{
    localhost.AuthorsService ws =new localhost.AuthorsService();
    ws.set_Credentials(System.Net.CredentialCache.get_DefaultCredentials());
    AuthorData.Merge(ws.GetAuthors());
}
```

2. En el menú **Ver**, haga clic en **Diseñador**. Haga doble clic en el botón **SaveData** para crear un controlador de eventos vacío para el evento **Click**.

Si existen cambios en el conjunto de datos, se crea un conjunto de datos nuevo del tipo **authors1** para guardar sólo los datos modificados. Este conjunto de datos se pasa luego al método **UpdateAuthors** del servicio Web XML. Se devuelve el conjunto de datos con los cambios aceptados y se actualiza el conjunto de datos **AuthorData** para reflejar los cambios nuevos. Para obtener más información sobre cómo aceptar cambios en un conjunto de datos, vea [Introducción a los adaptadores de datos](#).

**Nota** En una aplicación de producción, habría que considerar cuestiones de concurrencia de datos en este paso. Para obtener más información, vea [Introducción a la concurrencia de datos en ADO.NET](#).

Agregue el código siguiente a los métodos:

```
// Visual J#
private void SaveData_Click (Object sender, System.EventArgs e)
{
    if (AuthorData.HasChanges())
    {
        localhost.AuthorsService ws = new localhost.AuthorsService();
        ws.set_Credentials(System.Net.CredentialCache.get_DefaultCredentials());
        localhost.authors1 diffAuthors = new localhost.authors1();
        diffAuthors.Merge(AuthorData.GetChanges());
        diffAuthors = ws.UpdateAuthors(diffAuthors);
        AuthorData.Merge(diffAuthors);
    }
}
```



## Para ejecutar la aplicación

1. En el menú **Archivo**, elija **Guardar todo**.
2. Seleccione **AuthorsWinClient** en el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* (ratón) y después elija **Establecer como proyecto de inicio**.
3. Presione CTRL+F5 para ejecutar la aplicación.

Se muestra una ventana que contiene una tabla vacía con encabezados desde la tabla **authors** en la base de datos pubs.

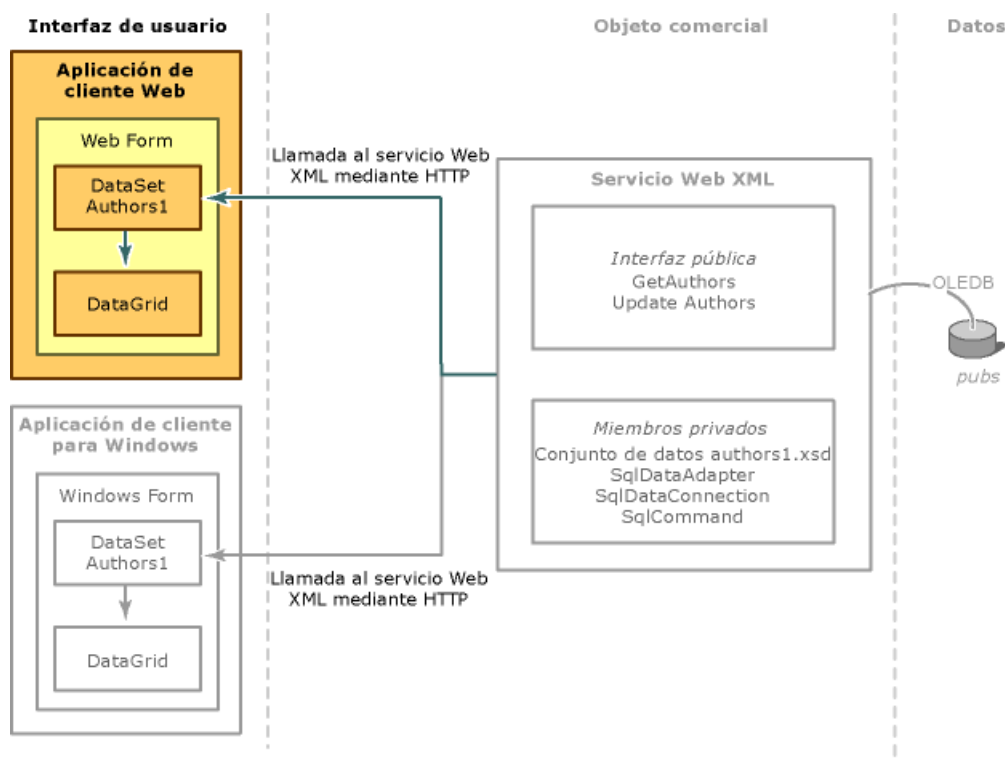
4. Haga clic en **Load** para rellenar la tabla, realice algunos cambios y después haga clic en **Save** para guardar los cambios realizados.

En la sección anterior, se agregó un proyecto de formularios Windows Forms a su solución para actuar como una interfaz de Windows. Se conectaron los formularios Windows Forms al servicio Web XML que se creó en la primera sección y se utilizaron los controles **DataGrid** y **Button** para crear la interfaz de usuario para cargar y actualizar los datos. En la siguiente sección, se creará una interfaz de usuario basada en Web para la solución.

## Opción de interfaz de usuario Web

Una interfaz Web ayuda a que la aplicación llegue a una amplia variedad de equipos cliente y exploradores. Todo el proceso de la interfaz de usuario se realiza en el servidor Web y no en el cliente. Al utilizar una página de formularios Web Forms, se creará una interfaz Web que tiene acceso al mismo objeto comercial de nivel medio que la interfaz de Windows que se creó en la sección anterior. El siguiente diagrama resalta el cliente que se implementará.

### Cliente Web en la aplicación distribuida



## Para crear la aplicación de formularios Web Forms

1. En el menú **Archivo**, elija **Agregar proyecto** y, a continuación, haga clic en **Nuevo proyecto**.
2. En el cuadro de diálogo **Agregar nuevo proyecto**, seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y, a continuación, seleccione **Aplicación Web ASP.NET** en el panel **Plantillas**.
3. En el cuadro **Ubicación**, escriba el nombre del servidor Web con el nombre del proyecto, **http://NombreDeServidor/AuthorsWebClient** y, después, haga clic en **Aceptar**.

Se agregará un nuevo proyecto al Explorador de soluciones. Una página de formularios Web Forms denominada **WebForm1.aspx** se agrega al proyecto y se carga dentro del diseñador.


4. Agregue una referencia Web al proyecto Servicio Web ASP.NET que creó anteriormente:
  - a. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el proyecto **AuthorsWebClient** y después haga clic en **Agregar referencia Web** en el menú contextual.
  - b. Introduzca la ubicación del archivo **.asmx** del proyecto de servicio Web XML

(<http://NombreDeServidor/AuthorsWebService/AuthorsService.aspx>) en el cuadro **Dirección** de la parte superior del cuadro de diálogo **Agregar referencia Web** y, a continuación, presione ENTRAR.

c. Haga clic en **Agregar referencia**.

d. Ahora puede crear una instancia del conjunto de datos **authors1** en la aplicación.

### Para agregar controles a una página Web

1. Arrastre un objeto **DataSet** desde la ficha **Datos** del Cuadro de herramientas hasta el formulario. Aparecerá el cuadro de diálogo **Agregar conjunto de datos**. Seleccione **Conjunto de datos con tipo** y elija "AuthorsWinClient.NombreDeServidor.authors1" de la lista **Nombre**. Se agregará un objeto **DataSet** a la bandeja de componentes.
2. Seleccione el objeto **DataSet** y establezca la propiedad **Name** en **AuthorData**.
3. Arrastre un control **DataGrid** desde la ficha **Web Forms** del Cuadro de herramientas hasta el formulario.
4. En la ventana **Propiedades** para el control **DataGrid**, establezca la propiedad **DataSource** en **AuthorData** y la propiedad **DataMember** en **authors**. Estos valores de configuración aparecen en las listas desplegables de estos controles. Los encabezados de columna del **DataGrid** se establecen en los nombres de columna de la tabla **authors**.
5. Para admitir la edición en contexto en el control **DataGrid**, se necesita agregar una columna **Editar, Actualizar, Cancelar**, que contendrá un botón **Editar**. Cuando el usuario haga clic en el botón **Editar**, el contenido de la fila se mostrará en cuadros de texto y el botón **Editar** se reemplazará por los botones **Actualizar** y **Cancelar**. Agregar esta columna:
  - a. Haga clic en el vínculo **Generador de propiedades** en la parte inferior de la ventana **Propiedades** y después seleccione la ficha **Columnas** en el cuadro de diálogo.
  - b. Expanda el elemento **Columnas Botón** en el panel **Columnas disponibles**.
  - c. Seleccione **Editar, Actualizar, Cancelar** y haga clic en el botón Agregar ().
  - d. Haga clic en **Aceptar**.

### Para agregar código para los botones Editar, Actualizar y Cancelar

1. Haga clic con el botón secundario en el formulario y seleccione **Ver código**. Agregar código al evento **Page\_Load** para rellenar el control **DataGrid**, como se muestra a continuación. Este código crea una instancia del servicio Web XML, rellena el conjunto de datos **AuthorData** y enlaza el conjunto de datos al control **DataGrid**. Cada vez que se devuelve la página al usuario, la cuadrícula contendrá una copia reciente de los datos de la base de datos. La propiedad **Credentials** del servicio Web XML se utiliza para pasar la identidad al servicio Web XML, que a su vez la pasa al servidor de la base de datos.

**Nota** En este tutorial, se recupera una copia nueva de la tabla **authors** de la base de datos cada vez que la página de formularios Web Forms se descarga en el explorador cliente. Esto podría proporcionar un rendimiento óptimo. Para conocer otros métodos de rendimiento óptimo, vea [Recomendaciones sobre la estrategia de acceso a datos Web](#).

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    localhost.AuthorsService ws = new localhost.AuthorsService();
    ws.set_Credentials(System.Net.CredentialCache.get_DefaultCredentials());
    AuthorData.Merge(ws.GetAuthors());
    if (! get_IsPostBack())
    {
        DataGrid1.DataBind();
    }
}
```

2. Cuando un usuario hace clic en el botón **Editar**, se produce el evento **EditCommand** del control **DataGrid**. Utilice este evento para cambiar la propiedad **EditItemIndex** del control **DataGrid**. Se mostrará la fila especificada por la propiedad **EditItemIndex** con todos los datos en cuadros de texto.
  - a. Cree un controlador de eventos para el evento **EditCommand**.

En Visual J#, haga clic en el control DataGrid1. En la ventana Propiedades, haga clic en el botón Eventos para mostrar la lista de los eventos **DataGrid**. Haga doble clic en el evento **EditCommand**.
  - b. Agregue el código que se muestra a continuación.

```
// Visual J#
private void DataGrid1_EditCommand (Object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    DataGrid1.set_EditItemIndex( e.get_Item().get_ItemIndex());
    DataGrid1.DataBind();
}
```

3. Cree un controlador de eventos para el evento **CancelCommand**. (Vea el paso 2 para agregar controladores de eventos.)

Cuando se hace clic en el botón **Cancelar**, se produce el evento **CancelCommand** del control **DataGrid**. Utilice este evento para establecer la propiedad **EditItemIndex** en **-1** para que así se muestre de nuevo la fila actual como texto. Agregue el código que se muestra a continuación.

```
// Visual J#
private void DataGrid1_CancelCommand (Object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    DataGrid1.set_EditItemIndex(-1);
    DataGrid1.DataBind();
}
```

4. Cuando se hace clic en el botón **Actualizar**, se produce el evento **UpdateCommand** del control **DataGrid**. Este método es responsable de la actualización del conjunto de datos **AuthorData** con los cambios procedentes del control **DataGrid** y de la propagación de estos cambios a la base de datos a través del servicio Web XML. Cree un controlador de eventos para el evento **UpdateCommand**. (Vea el paso 2 para agregar controladores de eventos.) Agregue el código que se muestra a continuación.

```
// Visual J#
private void DataGrid1_UpdateCommand (Object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    // Change the data in the dataset.
    for (int i=1; i<AuthorData.get_authors().get_Columns().get_Count(); i++)
    {
        TextBox t = (TextBox)((TableCell)(e.get_Item().get_Cells().get_Item(i))).get_Controls().get_Item(0);
        DataRow row = AuthorData.get_authors().get_Item(e.get_Item().get_DataSetIndex());
        row.set_Item(AuthorData.get_authors().get_Columns().get_Item(i-1).get_Caption(),t.get_Text());
    }

    // Update the database.
    if (AuthorData.HasChanges())
    {
        localhost.AuthorsService ws = new localhost.AuthorsService();
        ws.set_Credentials(System.Net.CredentialCache.get_DefaultCredentials());
        localhost.authors1 diffAuthors = new localhost.authors1();
        diffAuthors.Merge(AuthorData.GetChanges());
        ws.UpdateAuthors(diffAuthors);
        AuthorData.Merge(diffAuthors);
    }
    DataGrid1.set_EditItemIndex( -1);
    DataGrid1.DataBind();
}
```

Es necesario configurar los valores de seguridad del proyecto para trabajar con seguridad integrada. Para ello se desactiva el acceso anónimo y se activa la suplantación. Para obtener más información, vea [Modelo de seguridad](#).

## Para configurar la autenticación de Windows integrada

Para configurar la autenticación de Windows integrada para el proyecto, es necesario cambiar los archivos del proyecto y configurar el proyecto utilizando la herramienta **Servicios de Internet Information Server**.

1. Inicie la herramienta **Servicios de Internet Information Server**. Se puede ejecutar desde **Herramientas administrativas** en el **Panel de control**. (Para obtener más información acerca de esta herramienta, vea la documentación de ayuda de Windows.) Expanda el nodo de su servidor.
2. Expanda el nodo **Sitio Web predeterminado**.
3. Haga clic con el botón secundario del *mouse* (ratón) en el nodo de **AuthorsWebClient** y elija **Propiedades** en el menú contextual.
4. Haga clic en la ficha **Seguridad de directorios**.
5. Haga clic en el botón **Editar** en la sección **Control de autenticación y acceso anónimo**.
6. Desactive la casilla de verificación **Acceso anónimo**.
7. Active la casilla **Autenticación de Windows integrada**. Acaba de configurar el directorio del servicio Web XML.
8. Al volver al proyecto en Visual Studio, haga doble clic en el archivo Web.config en el Explorador de soluciones.
9. Agregue la siguiente etiqueta en la línea después de la etiqueta `<system.web>` para configurar la seguridad integrada del servicio Web XML.

```
<identity impersonate="true"/>
```

## Para ejecutar la aplicación

1. Seleccione **AuthorsWebClient** en el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* y luego elija **Establecer como proyecto de inicio**.
2. Presione CTRL+F5 para ejecutar la aplicación.

En esta sección, se agregó un proyecto Aplicación Web ASP.NET a la solución para actuar como una interfaz de exploración para los datos. Se conectó la página de formularios Web Forms al servicio Web XML que se creó en la primera sección y se utilizó un control **DataGrid** para crear la interfaz de usuario para mostrar y editar los datos.

## Implementar la solución

El siguiente paso sería implementar la aplicación Web en un servidor y crear un instalador para la aplicación para Windows. Para obtener más información sobre la implementación de una aplicación Web, vea [Tutorial: implementar una solución Web](#). Para obtener más información sobre la implementación de una aplicación para Windows, vea [Tutorial: implementar una aplicación para Windows](#).

## Vea también

Programar el Web con servicios Web XML | [Páginas de formularios Web Forms](#) | [Formularios Windows Forms](#) | [Servicios Web XML en código administrado](#) | [Introducción a las aplicaciones distribuidas y a la integración de datos](#)

# Data Walkthroughs

These step-by-step topics cover common data and XML scenarios.

**Tip** If you are new to ADO.NET, try one of the Windows Forms or Web Forms data walkthroughs before trying one of the general data walkthroughs.

## In the Visual Basic and Visual C# Documentation

### Windows Forms Data-Access Walkthroughs

#### [Simple Data Access in a Windows Form](#)

Describes the basic steps required for accessing SQL Server data via a dataset in a read-write Windows Form.

#### [Displaying Data in a Windows Form Using a Parameterized Query](#)

Describes how to create a dataset containing only selected records, based on criteria that users provide at run time in a Windows Form.

#### [Creating a Master-Detail Windows Form](#)

Describes how to create a parent-child relationship in a dataset and display related records in a Windows Form.

### Web Forms Data-Access Walkthroughs

#### [Displaying Data in a Web Forms Page](#)

Illustrates a basic data-access scenario, providing step-by-step instructions for how to create a Web Forms page that displays data in a grid control.

#### [Creating Read-Only Data Access in a Web Forms Page](#)

Describes how to use an ADO.NET data command and data reader to provide optimized access to read-only data in a Web Forms page.

#### [Updating Data Using a Database Update Query in Web Forms](#)

Describes the basic techniques of creating a Web Forms page that uses a data command to update the database.

#### [Using a DataGrid Web Control to Read and Write Data](#)

Gives step-by-step details on how to use a grid control to allow users to view and edit data.

#### [Creating Paged Data Access Using a Web Forms Page](#)

Describes how to create a Web Forms page that allows users to view data records a few records at a time, paging back and forth to view earlier and later records.

### General Data Walkthroughs

#### [Creating a Distributed Application](#)

Illustrates how to create an application with a middle-tier component (an XML Web service) that is used to get and update data.

#### [Creating a Dataset with Tables, Keys, and One-to-Many Relationships](#)

Describes the steps to create a dataset and use it to validate data in a one-to-many or keyed relationship.

#### [Handling a Concurrency Exception](#)

Describes how to use the **DBConcurrencyException** object to identify concurrency exceptions and the actual record that caused the error.

#### [Mapping Data Source Tables to Dataset Tables](#)

Describes how to load data from a database into a dataset based on a different schema using data adapter table and column mappings.

### XML Data and XML Schema Walkthroughs

#### [Creating an XML File with an Associated XML Schema](#)

Describes how to create an XML file and a schema, associating the two, and working on the XML file in the designer's XML view.

#### [Creating an XML Schema with the XML Designer](#)

Describes the steps to create an XML purchase order schema as part of a Windows application.

#### [Reading XML Data into a Dataset](#)

Shows how to load XML data into a dataset, display it in a Windows Forms **DataGrid** control, then display an XML Schema for the XML file in a text box.

#### [Displaying an XML Document in a Web Forms Page using Transformations](#)

Gives step-by-step details on how to read an XML file and display it in different ways on a Web Forms page.

### Additional Information

### [Samples and Walkthroughs](#)

Provides access to all of the sample applications and step-by-step topics in Visual Studio.

### [Accessing Data](#)

Provides links to information about working with ADO.NET data in Visual Basic and Visual C#.

### [Walkthroughs in Visual Basic and Visual C#](#)

Shows a complete list of walkthroughs related to managed development in Visual Basic and Visual C#.

### [Visual Basic Code Example Topics](#)

Shows quickstart-like code examples that help you perform common tasks in Visual Basic .NET.

# Tutorial: crear un conjunto de datos con tablas, claves y relaciones uno a varios con Visual J#

En este tutorial creará un conjunto de datos para validar datos en una relación uno a varios (una relación uno a varios también puede denominarse una *relación con claves*). Le guiará a través del proceso de creación de un proyecto de Visual J# y agregar un nuevo elemento Dataset al mismo. A continuación, creará elementos de tipo complejo que representan tablas de datos con columnas. Aprenderá a definir claves principales en tablas. Finalmente, aprenderá a crear relaciones entre tablas basándose en las claves definidas. Cuando haya finalizado, podrá examinar el código XML subyacente que se creó mediante el Diseñador XML, para entender mejor cómo expresan los conjuntos de datos las relaciones con claves en XML nativo.

En este tutorial se incluye la creación de tablas, claves principales y relaciones, y consta de las tres secciones principales que se describen a continuación:

1. **Agregar un elemento Dataset a un proyecto y crear tres tablas.**
  - a. Crear una tabla Customers.
  - b. Crear una tabla Orders.
  - c. Crear una tabla OrderDetails.
2. **Crear dos claves principales.**
  - a. Crear una clave principal en la tabla Customers.
  - b. Crear una clave principal en la tabla Orders.
3. **Crear dos relaciones (objetos DataRelation).**
  - a. Crear una relación entre la tabla Customers (uno) y la tabla Orders (varios).
  - b. Crear una relación entre la tabla Orders (uno) y la tabla OrderDetails (varios).

## Crear un nuevo proyecto de aplicación para Windows y agregarle un nuevo elemento Dataset

### Para crear y nombrar un nuevo proyecto

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows**.
3. Asigne al proyecto el nombre **KeyedRelationshipWalkthrough** y, a continuación, haga clic en **Aceptar** para crear el proyecto.

Ya que este tutorial requiere un conjunto de datos, debe agregar uno al proyecto.

### Para agregar un nuevo elemento Dataset al proyecto

1. En el menú **Proyecto**, elija **Agregar nuevo elemento**.  
Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.
2. En el área Plantillas del cuadro de diálogo **Agregar nuevo elemento**, seleccione **Dataset**.
3. Asigne al conjunto de datos el nombre **CustomerOrders** y haga clic en **Abrir**.  
Visual Studio agregará un archivo denominado **CustomerOrders.xsd** y **CustomerOrders.jsl** al proyecto y lo cargará automáticamente en el diseñador.
4. En el menú **Proyecto**, si aún no lo ha hecho, seleccione **Mostrar todos los archivos**. Habrá un borde visible alrededor del icono cuando se haya seleccionado.
5. En el Explorador de soluciones, expanda el nodo **CustomerOrders.xsd**.

Ahora puede ver el archivo **CustomerOrders.jsl** que se creó en el paso 3. Éste es el conjunto de datos real o, más exactamente, una clase de conjunto de datos. El archivo **CustomerOrders.xsd** es un esquema XML que definirá la estructura del conjunto de datos.

## Crear tres tablas en el conjunto de datos

En esta sección, se explica cómo agregar elementos al conjunto de datos que representarán tablas de datos.

### Para crear la tabla Customers

1. En la ficha **Esquema XML** del Cuadro de herramientas, seleccione un **element** y arrástrelo hasta el diseñador.
2. En la celda derecha de la parte superior de la '**E**', resalte el nombre predeterminado **element1** y escriba **Customers**.
3. Mediante la tecla TABULADOR vaya a la celda central de la fila siguiente y escriba **CustomerID**. El tipo de datos para este elemento es string, la opción predeterminada, así que puede dejarlo así. No obstante, si necesita asignar un tipo de datos que no sea string, puede utilizar la tecla TABULADOR para ir a la lista desplegable situada a la derecha del elemento deseado y seleccionar el tipo de datos adecuado.

**Nota** Observe también cómo ha pasado de la pequeña celda de la celda que indicaba **element1**. Ésta es la celda donde se eligen otros tipos de elementos, como por ejemplo los atributos. Dado que el valor predeterminado del elemento es **E**, puede dejar el valor como está.

4. Repita el paso 3 para crear nuevas filas en el elemento **Customers** para lo siguiente:

Nombre del elemento	Tipo de datos
<b>CompanyName</b>	<b>string</b>
<b>ContactName</b>	<b>string</b>
<b>ContactTitle</b>	<b>string</b>
<b>Address</b>	<b>string</b>
<b>City</b>	<b>string</b>
<b>Region</b>	<b>string</b>
<b>PostalCode</b>	<b>string</b>

#### Para crear la tabla **Orders**

1. En la ficha **Esquema XML** del Cuadro de herramientas, seleccione un **element** y arrástrelo hasta el diseñador.
2. En la celda derecha de la parte superior de la '**E**', resalte el nombre predeterminado **element1** y escriba **Orders**.
3. Mediante la tecla TABULADOR vaya a la celda central de la fila siguiente, escriba **OrderID** y establezca el tipo de datos en string.
4. Repita el paso 3 y cree nuevas filas en el elemento **Orders** para lo siguiente:

Nombre del elemento	Tipo de datos
<b>CustomerID</b>	<b>string</b>
<b>OrderDate</b>	<b>date</b>
<b>ShippedDate</b>	<b>date</b>
<b>ShipVia</b>	<b>string</b>

#### Para crear la tabla **OrderDetails**

1. En la ficha **Esquema XML** del Cuadro de herramientas, seleccione un **element** y arrástrelo hasta el diseñador.
2. En la celda derecha de la parte superior de la '**E**', resalte el nombre predeterminado **element1** y escriba **OrderDetails**.
3. Mediante la tecla TABULADOR vaya a la celda central de la fila siguiente, escriba **OrderID** y establezca el tipo de datos en string.
4. Repita el paso 3 y cree nuevas filas en el elemento **OrderDetails** para lo siguiente:

Nombre del elemento	Tipo de datos
<b>ProductID</b>	<b>integer</b>
<b>UnitPrice</b>	<b>decimal</b>
<b>Quantity</b>	<b>short</b>

### Crear las dos claves principales en las tablas

En esta sección se explica cómo designar columnas en las tablas que ha creado en la sección anterior como claves principales. Para crear relaciones debe tener al menos una clave definida en la tabla primaria (la tabla primaria es la tabla que representa el lado 'uno' de una relación uno a varios).

#### Para crear la clave principal en la tabla **Customers**

1. En la tabla **Customers**, seleccione la fila **CustomerID** haciendo clic a la izquierda de la **E**.
2. Haga clic con el botón secundario del *mouse*, elija **Agregar** y seleccione **Nueva clave** en el menú contextual.

Aparece el cuadro de diálogo **Editar clave**.

3. Cambie el nombre de la clave a **CustomersIDKey**.



- El cuadro de lista desplegable **Elemento** se establecerá en **Customers**, que indica que se encuentra en la tabla Customers.

**Nota** La tabla **Customers** realmente es el elemento **Customers** que está definido como tipo complejo; es un elemento de tipo complejo que representa a las tablas en conjuntos de datos y esquemas XML. Para obtener más información, vea [Crear relaciones uno a varios en conjuntos de datos y esquemas XML](#).

- En el área **Campos**, seleccione **CustomerID** en la lista desplegable. Si hizo clic con el botón secundario del *mouse* en la fila **CustomerID** para agregar la clave, este valor debe ser el predeterminado. Si hizo clic en una fila diferente, deberá seleccionar el elemento **CustomerID**.

**Nota** El área **Campos** es donde se selecciona qué elemento (de la definición de tipo complejo del elemento de tabla) se desea definir como clave.

- Active la casilla de verificación **Clave principal del conjunto de datos**, para definir esta clave como clave principal.

**Nota** Si no activó la casilla de verificación **Clave principal del conjunto de datos**, la clave se definirá como clave única, en lugar de como clave principal.

- Haga clic en **Aceptar** para cerrar el cuadro de diálogo **Editar clave**.

En la fila **CustomerID** se coloca un icono de clave para identificarla como clave.

### Para crear la clave principal en la tabla Orders

- En la tabla **Orders**, seleccione la fila **OrderID** haciendo clic a la izquierda de la **E**.
- Haga clic con el botón secundario del *mouse* en la fila seleccionada, elija **Agregar** y seleccione **Nueva clave** en el menú contextual.

Aparece el cuadro de diálogo **Editar clave**.

- Cambie el nombre de la clave a **OrdersIDKey**.
- El cuadro de lista desplegable **Elemento** se establecerá en **Orders**, que indica que se encuentra en la tabla **Orders**.
- En el área **Campos**, seleccione **OrderID** en la lista desplegable. Si hizo clic con el botón secundario del *mouse* en la fila **OrderID** para agregar la clave, este valor debe ser el predeterminado. Si hizo clic en una fila diferente, deberá seleccionar el elemento correcto.
- Active la casilla de verificación **Clave principal del conjunto de datos** para definir esta clave como clave principal.
- Haga clic en **Aceptar** para cerrar el cuadro de diálogo **Editar clave**.

En la fila **OrderID** se coloca un icono de clave para identificarla como clave.

Ahora tiene definidas las claves que se utilizarán para definir las relaciones entre tablas. Quizá se pregunte por qué no se definió una clave en la tabla **OrderDetails**. Se ha dicho al comienzo de esta sección que "para crear relaciones debe tener al menos una clave definida en la tabla primaria (la tabla primaria es la tabla que representa el lado 'uno' de una relación uno a varios)". La tabla **OrderDetails** no representa el lado 'uno' de ninguna relación de este conjunto de datos; por tanto, no se requiere ninguna definición de clave.

### Crear las dos relaciones entre las tablas

Una relación es la asociación entre una columna con claves de una tabla y muchos registros que tienen una columna asociada en otra tabla. En esquemas XML, las relaciones se definen con el elemento `keyref`. Esta sección explica cómo definir estas relaciones.

#### Para crear una relación (objeto DataRelation) entre la tabla Customers y la tabla Orders

- Haga clic con el botón secundario del *mouse* en cualquier punto de la tabla **Orders**, elija **Agregar** y seleccione **Nueva relación** en el menú contextual.

Aparece el cuadro de diálogo **Editar relación**.

**Nota** Asegúrese de que agrega la nueva relación a la tabla que representa el lado varios de una relación uno a varios (en este caso, agregue la relación a la tabla **Orders**).

- El cuadro de diálogo **Editar relación** asigna un nombre predeterminado de **CustomersOrders**. Puede dejarlo así.
- El **Elemento primario** debería establecerse en **Customers**. El elemento primario es el elemento (o tabla) que representa el lado 'uno' en una relación uno a varios. En este caso, la tabla **Customers** representa el lado 'uno'.
- El **Elemento secundario** debe establecerse en **Orders**. El elemento secundario es el elemento (o tabla) que representa el

lado 'varios' en una relación uno a varios. En este caso, la tabla **Orders** representa el lado 'varios'.

5. La clave debería establecerse en **CustomersIDKey**, que se ha definido anteriormente.

**Nota** Al crear una relación uno a varios la definición de clave necesita estar en la tabla **primaria**, porque es la tabla que contiene el valor único.

6. Los campos de clave son de sólo lectura y contienen la columna o columnas de clave definidas en la definición de clave.

**Nota** Las claves compuestas son definiciones de clave en las que varias columnas de una tabla forman parte de la definición de clave. Si una clave compuesta se ha definido en **CustomersKey1**, todas las columnas (o campos) estarían enumeradas en el área de campos de clave. Este tutorial no incluye claves compuestas.

7. El área **Campos de clave externa** es donde se selecciona el campo de la tabla secundaria que es equivalente a la clave de la tabla **primaria**. Este campo debería establecerse a **CustomerID** para que coincida con el campo clave de la tabla primaria.
8. Haga clic en **Aceptar** para aceptar los restantes valores predeterminados.

Se creará un objeto **DataRelation** y en la superficie del diseñador aparecerá una representación visual.

### Para crear una relación (objeto **DataRelation**) entre la tabla **Orders** y la tabla **OrderDetails**

1. Haga clic con el botón secundario del *mouse* en cualquier punto de la tabla **OrderDetails**, elija **Agregar** y seleccione **Nueva relación** en el menú contextual.

Aparece el cuadro de diálogo **Editar relación**.

**Nota** Asegúrese de que agrega la nueva relación a la tabla que representa el lado varios de una relación uno a varios (en este caso, agregue la relación a la tabla **OrderDetails**).

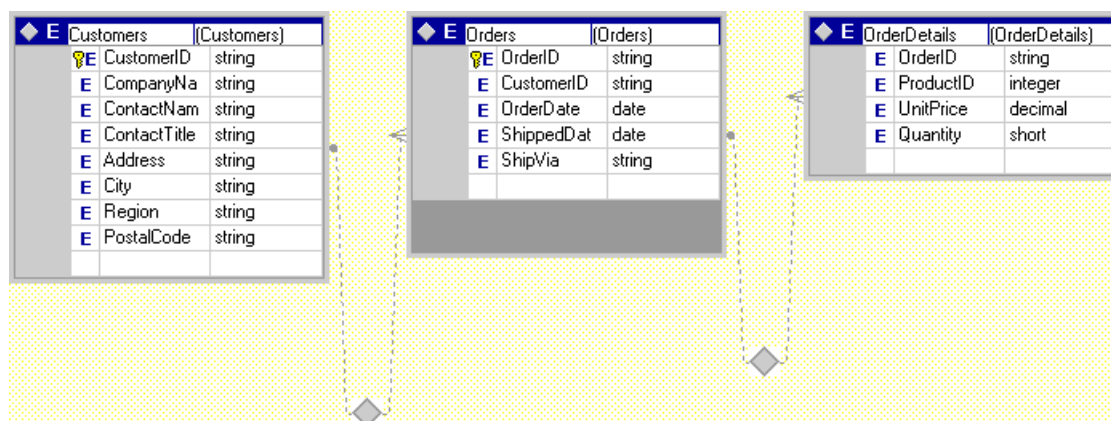
2. Cambie el nombre predeterminado de **CustomersOrderDetails** a **OrdersOrderDetails** para reflejar las tablas adecuadas en la relación.
3. El **Elemento primario** debe establecerse en **Orders**.
4. El **Elemento secundario** debe establecerse en **OrderDetails**.
5. La **Clave** debería establecerse en **OrdersIDKey**, que se ha definido anteriormente.

Los campos de clave son de sólo lectura y contienen la columna o columnas de clave definidas en la definición de clave seleccionada en el paso 5.

6. El área **Campos de clave externa** es donde se selecciona el campo de la tabla secundaria que es equivalente a la clave de la tabla **primaria**. Compruebe que este campo está establecido en **OrderID** para que coincida con el campo clave de la tabla primaria.
7. Haga clic en **Aceptar** para aceptar los restantes valores predeterminados.

Se creará un objeto **DataRelation** y en la superficie del diseñador aparecerá una representación visual.

Ésta debería ser la apariencia del conjunto de datos en el Diseñador XML:



Acaba de crear tres tablas relacionadas en un conjunto de datos. Si cambia a la vista XML, el código deberá ser equivalente al siguiente. Lo importante es observar cómo se estructura el código XML nativo en un modelo relacional uno a varios (o con claves). Como se puede observar, todas las tablas **Customers**, **Orders** y **OrderDetails** se encuentran jerárquicamente al mismo nivel, como tablas secundarias directas del elemento **CustomerOrders**. El elemento **CustomerOrders** es el que representa el

conjunto de datos.

**Nota** Observe cómo las claves están definidas jerárquicamente como secundarias del elemento **CustomerOrders**. Es muy importante comprender por qué las definiciones de claves no se presentan como secundarias anidadas en el elemento que representa la tabla donde está definida la clave. Si éste fuera el caso, la clave sería exclusiva sólo para cada registro, no para todos los registros de la tabla. Mediante la definición de la clave en el propio conjunto de datos (el elemento **CustomerOrders**), la clave controla la exclusividad en todo el conjunto de datos.

```
<xs:schema id="CustomerOrders" targetNamespace="http://tempuri.org/CustomerOrders.xsd" elementFormDefault="qualified" xmlns="http://tempuri.org/CustomerOrders.xsd" xmlns:mstns="http://tempuri.org/CustomerOrders.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="CustomerOrders" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
              <xs:element name="CompanyName" type="xs:string" minOccurs="0" />
              <xs:element name="ContactName" type="xs:string" minOccurs="0" />
              <xs:element name="ContactTitle" type="xs:string" minOccurs="0" />
              <xs:element name="Address" type="xs:string" minOccurs="0" />
              <xs:element name="City" type="xs:string" minOccurs="0" />
              <xs:element name="Region" type="xs:string" minOccurs="0" />
              <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Orders">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="OrderID" type="xs:string" minOccurs="0" />
              <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
              <xs:element name="OrderDate" type="xs:date" minOccurs="0" />
              <xs:element name="ShippedDate" type="xs:date" minOccurs="0" />
              <xs:element name="ShipVia" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="OrderDetails">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="OrderID" type="xs:string" minOccurs="0" />
              <xs:element name="ProductID" type="xs:integer" minOccurs="0" />
              <xs:element name="UnitPrice" type="xs:decimal" minOccurs="0" />
              <xs:element name="Quantity" type="xs:short" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    <xs:key name="CustomersIDKey" msdata:PrimaryKey="true">
      <xs:selector xpath="./mstns:Customers" />
      <xs:field xpath="mstns:CustomerID" />
    </xs:key>
    <xs:key name="OrdersIDKey" msdata:PrimaryKey="true">
      <xs:selector xpath="./mstns:Orders" />
      <xs:field xpath="mstns:OrderID" />
    </xs:key>
    <xs:keyref name="CustomersOrders" refer="CustomersIDKey">
      <xs:selector xpath="./mstns:Orders" />
      <xs:field xpath="mstns:CustomerID" />
    </xs:keyref>
    <xs:keyref name="OrdersOrderDetails" refer="OrdersIDKey">
      <xs:selector xpath="./mstns:OrderDetails" />
      <xs:field xpath="mstns:OrderID" />
    </xs:keyref>
  </xs:element>
</xs:schema>
```

```
</xs:keyref> </xs:element>  
</xs:schema>
```

## **Vea también**

[Crear relaciones uno a varios en conjuntos de datos y esquemas XML](#) | [Datos relacionales en esquemas XML](#) | [Crear tablas en esquemas XML](#) | [Crear claves principales y únicas en esquemas XML](#) | [Eliminar claves principales y únicas](#) | [Crear objetos DataRelation con el Diseñador XML](#) | [Introducción a conjuntos de datos](#) | [Relaciones en los conjuntos de datos de ADO.NET](#) | [Tutoriales sobre datos](#)

# Tutorial: controlar una excepción de concurrencia con Visual J#

En este tutorial, se creará una aplicación para Windows que provoca un error de concurrencia y muestra una estrategia para controlarlo. El tutorial simula que hay dos usuarios trabajando con los mismos datos al mismo tiempo. El formulario Windows Form que va a crear le permite actuar como ambos usuarios desde un solo formulario.

Este tutorial demostrará las tareas siguientes:

1. El Usuario 1 y el Usuario 2 llenan sus respectivos conjuntos de datos con los mismos datos.
2. El Usuario 2 modifica un registro, actualiza el conjunto de datos y escribe los cambios en el origen de datos.
3. El Usuario 1 modifica el mismo registro, actualiza el conjunto de datos e intenta escribir los cambios en el origen de datos, pero se genera un error de concurrencia.

Deberá detectar el error, y a continuación, mostrar las distintas versiones del registro, lo que permitirá al usuario (usted) determinar qué sucedería con los cambios pendientes realizados por Usuario 1.

**Nota** Usar un conjunto de datos es sólo una de las opciones para obtener acceso a datos, y no es la elección ideal en algunos escenarios. Sin embargo, los conjuntos de datos suelen ser la elección adecuada en las aplicaciones de formularios Windows Forms; además, en este tutorial se ilustra un escenario en el que los conjuntos de datos son la opción apropiada. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos](#).

Requisitos previos de este tutorial:

- Acceso a la base de datos de ejemplo Pubs de SQL Server con permiso para realizar actualizaciones.

## Crear un nuevo proyecto y una nueva conexión de datos

El tutorial comienza con la creación de una nueva aplicación para Windows en Visual J#.

### Para crear un nuevo proyecto

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows**.
3. Asigne al proyecto el nombre **concurrency\_walkthrough** y haga clic en **Aceptar**.

Visual Studio agregará el proyecto concurrency\_walkthrough al Explorador de soluciones y mostrará un nuevo formulario Windows Forms en el diseñador.

### Para crear una nueva conexión de datos

1. En el Explorador de servidores, cree una nueva conexión con la base de datos de ejemplo Pubs. Para obtener más información, vea [Agregar nuevas conexiones de datos en el Explorador de servidores](#).
2. En el Explorador de servidores, expanda la conexión creada en el paso anterior.
3. Expanda el área **Tablas**.
4. Arrastre la tabla **authors** hasta su formulario.

En la bandeja de componentes, debajo del formulario, aparecen un objeto **Connection** y un objeto **DataAdapter**.

## Crear los conjuntos de datos

En esta sección, creará dos conjuntos de datos denominados **DsAuthors1** y **DsAuthors2**. Estos conjuntos de datos representan los datos con los que trabajan los dos usuarios simultáneos. A continuación, agregará dos controles [DataGrid](#) al formulario y los enlazará a los conjuntos de datos. Finalmente, se agregarán dos controles [Button](#) al formulario: un botón **Update** y un botón **Reset**. El botón **Update** cambia un registro de **DsAuthors1** e intenta escribir el cambio en la base de datos. El botón **Reset** restablece el registro original en la base de datos de modo que el tutorial funcione si se ejecuta más de una vez.

### Para crear dos nuevos conjuntos de datos

1. Seleccione el objeto **DataAdapter**.
2. En el menú **Datos**, seleccione **Generar conjunto de datos**.  
Aparecerá el cuadro de diálogo **Generar conjunto de datos**.
3. Seleccione **Nuevo** y asigne al conjunto de datos el nombre **DsAuthors**.

Una instancia denominada **DsAuthors1** aparece en la bandeja de componentes.

4. En la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **DataSet** hasta el formulario.

Aparece el cuadro de diálogo **Agregar conjunto de datos**.

5. Confirme que la opción **Conjunto de datos con tipo** está seleccionada y que **concurrency\_walkthrough.DsAuthors** aparece en el cuadro **Nombre**.

Una instancia denominada **DsAuthors2** aparece en la bandeja de diseño de componentes.

## Enlazar los datos y agregar botones

Las cuadrículas de datos sólo se utilizan para mostrar los datos. No se deben modificar datos en las cuadrículas de datos durante este tutorial.

### Para agregar dos controles **DataGrid** al formulario

1. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un objeto **DataGrid** hasta la parte izquierda de su formulario.
2. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un objeto **DataGrid** hasta la parte derecha de su formulario.

### Para enlazar los conjuntos de datos a los controles **DataGrid**

1. Seleccione **DataGrid1** y establezca las propiedades siguientes en la ventana **Propiedades**:

Propiedad	Valor
<a href="#">DataSource</a>	<b>DsAuthors1</b>
<a href="#">DataMember</a>	<b>authors</b>
<a href="#">CaptionText</a>	<b>DsAuthors1</b>

2. Seleccione **DataGrid2** y establezca las propiedades siguientes en la ventana **Propiedades**:

Propiedad	Valor
<b>DataSource</b>	<b>DsAuthors2</b>
<b>DataMember</b>	<b>Authors</b>
<b>CaptionText</b>	<b>DsAuthors2</b>

### Para agregar los botones **Update** y **Reset** al formulario

1. En la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **Button** hasta el formulario y colóquelo encima de **DataGrid1**.
2. Seleccione el botón y después, en la ventana **Propiedades**, asigne al botón el nombre **btnUpdate** y establezca su propiedad **Text** en **Update**.
3. En la ficha **Windows Forms** del Cuadro de herramientas, arrastre un segundo objeto **Button** hasta el formulario y colóquelo encima de **DataGrid2**.
4. Seleccione el botón y después, en la ventana **Propiedades**, asigne al botón el nombre **btnReset** y establezca su propiedad **Text** en **Reset**.

## Restablecer la base de datos

Agregaré al formulario código que restablezca la base de datos con valores conocidos (por si desea ejecutar el tutorial más de una vez).

### Para agregar código que restablezca la base de datos

- Haga clic con el botón secundario en el formulario, elija **Ver código** en el menú contextual y, a continuación, inserte el código siguiente encima del constructor **Form1**:

```
// Visual J#
private void resetDatabase()
{
    // Fill the dsAuthors dataset with data.
    sqlDataAdapter1.Fill(dsAuthors1);
    // Reset the au_fname in the first row to "John".
    dsAuthors1.get_authors().get_Item(0).set_au_fname("John");
}
```

```
// Write the record back to the database.
sqlDataAdapter1.Update(dsAuthors1);
// Close the connection.
sqlConnection1.Close();
}
```

## Llenar los conjuntos de datos

En este paso, se llenan ambos conjuntos de datos con los mismos datos.

### Para agregar código que llene los conjuntos de datos desde la base de datos

- Inserte el código siguiente en el Editor de código:

```
// Visual J#
private void filltheDataSets()
{
    sqlDataAdapter1.Fill(dsAuthors1);
    sqlDataAdapter1.Fill(dsAuthors2);
    sqlConnection1.Close();
}
```

## Simular cambios de User 2

### Para agregar código que simule los cambios realizados por User 2

- Inserte el código siguiente en el Editor de código:

```
// Visual J#
private void user2changes()
{
    // Simulate a second user changing a record.
    dsAuthors2.get_authors().get_Item(0).set_au_fname("User 2");
    // Write it back to the database.
    sqlDataAdapter1.Update(dsAuthors2.GetChanges());
    // Refresh dsAuthors2 with the updated data.
    sqlDataAdapter1.Fill(dsAuthors2);
    // Close the connection..
    sqlConnection1.Close();
}
```

## Crear el controlador de eventos Form\_Load

### Para agregar código al evento Form\_Load para inicializar el tutorial

1. En la vista Diseño, haga doble clic en un área vacía del formulario para crear automáticamente el controlador de eventos **Form\_Load**.
2. Agregue código de modo que el controlador de eventos tenga el aspecto siguiente:

```
// Visual J#
private void Form1_Load(Object sender, System.EventArgs e)
{
    resetDatabase();
    filltheDataSets();
    user2changes();
}
```

3. Guarde el proyecto.

## Ejecutar la aplicación

1. Presione F5 para ejecutar la aplicación.

El formulario aparece con dos cuadrículas de datos llenas con datos de la tabla "authors" de la base de datos Pubs. El campo **au\_fname** del primer registro de DsAuthors1 debe ser **John**. El campo **au\_fname** del primer registro de **DsAuthors2** debe ser **User 2**.

El formulario tendrá una apariencia parecida a la siguiente:

The screenshot shows a Windows application window titled 'Form1'. At the top, there are two buttons: 'actualizar' on the left and 'Restablecer' on the right. Below these buttons are two data grids. The first grid, 'DsAuthors1', has columns 'au\_id', 'au\_lname', and 'au\_fname'. The second grid, 'DsAuthors2', also has columns 'au\_id', 'au\_lname', and 'au\_fname'. Both grids contain 12 rows of data. The first row of 'DsAuthors1' shows '172-32-1176', 'White', and 'John'. The first row of 'DsAuthors2' shows '172-32-1176', 'White', and 'User 2'.

DsAuthors1			DsAuthors2		
au_id	au_lname	au_fname	au_id	au_lname	au_fname
172-32-1176	White	John	172-32-1176	White	User 2
213-46-8915	Green	Margie	213-46-8915	Green	Margie
238-95-7766	Carson	Cheryl	238-95-7766	Carson	Cheryl
267-41-2394	Hunter	Anne	267-41-2394	Hunter	Anne
274-80-9391	Straight	Dean	274-80-9391	Straight	Dean
341-22-1782	Smith	Jim	341-22-1782	Smith	Jim
409-56-7008	Bennet	Abraham	409-56-7008	Bennet	Abraham
427-17-2319	Dull	Anna	427-17-2319	Dull	Anna
472-27-2349	Gringlesby	Burt	472-27-2349	Gringlesby	Burt
486-29-1786	Locksley	Charlene	486-29-1786	Locksley	Charlene
527-72-3246	Greene	Morminstar	527-72-3246	Greene	Morminstar

2. En el menú **Depurar**, seleccione **Detener depuración**.

## Actualizar el conjunto de datos y escribir los cambios en la base de datos

A continuación, escribirá código que actualice la base de datos con los cambios del conjunto de datos **DsAuthors1**. Si la actualización se realiza correctamente, se llama al método [AcceptChanges](#) del conjunto de datos y un cuadro de mensaje informa de que el proceso ha sido satisfactorio. Si, por cualquier motivo, la actualización no se realiza correctamente, se detecta el error y un cuadro de mensaje, cuyo título es el tipo de objeto de error, presenta el mensaje de error. Aunque este tutorial está diseñado para provocar un error de concurrencia, el cuadro de mensaje de actualización satisfactoria se muestra para completar el proceso.

**Nota** El bloque **Try...Catch** capturará cualquier error que se produzca. Más adelante en el tutorial se agregará una instrucción catch adicional para controlar específicamente el error de concurrencia.

### Para actualizar la base de datos

1. Llame al método [Update](#).
2. Cree un controlador de excepciones que muestre un cuadro de mensaje.

El código debería ser como el que sigue a continuación:

```
// Visual J#
private void updateDatabase()
{
    try
    {
        // Update the database with the changes from dsAuthors1.
        sqlDataAdapter1.Update(dsAuthors1.GetChanges());
        dsAuthors1.AcceptChanges();
        sqlConnection1.Close();
        MessageBox.Show("The update was successful.");
    }
    catch (System.Exception ex)
    {
        // Display information about update errors.
        MessageBox.Show(ex.get_Message(), ex.GetType().ToString());
    }
}
```

A continuación, debe agregar código que modifique la columna **au\_fname** del conjunto de datos **DsAuthors1**. Después, el código llama al procedimiento **updateDatabase** para intentar escribir el cambio en la base de datos. Dado que el Usuario 2 ya había modificado este valor, se produce un error de concurrencia.



## Para actualizar el conjunto de datos DsAuthors1

1. Haga doble clic en el botón **Update**.
2. Cree el controlador de eventos **btnUpdate\_Click**:

```
// Visual J#
private void btnUpdate_Click(Object sender, System.EventArgs e)
{
    // Change au_fname in the first row of dsAuthors1 to "User 1".
    dsAuthors1.get_authors().get_Item(0).set_au_fname("User 1");
    updateDatabase();
}
```

3. Guarde el proyecto.
4. Presione F5 para ejecutar la aplicación.
5. Haga clic en el botón **Update** para cambiar el campo **au\_fname** del primer registro a **User 1**.

Se provocará el error de concurrencia.

## Controlar errores de concurrencia

La manera de controlar el error depende de las reglas de empresa específicas por las que se rija la aplicación. La siguiente estrategia para controlar el error se utilizará a modo de ejemplo. La aplicación presentará al usuario tres versiones del registro:

- El registro actual de la base de datos
- El registro original del conjunto de datos
- Los cambios propuestos del conjunto de datos

Después, el usuario podrá sobrescribir la base de datos con el cambio propuesto o cancelar esta actualización y actualizar el conjunto de datos.

## Crear un controlador de errores personalizado

Cuando se lleva a cabo una actualización, normalmente hay que hacerla en un controlador de excepciones estructurado para poder detectar los errores. En el código que utilizó antes, todos los errores se detectaban mediante un bloque **try...catch**, es decir, una estructura que incluye una instrucción **catch** genérica para todo tipo de errores.

También se pueden detectar errores específicos para responder según corresponda. A modo de ejemplo, este tutorial agregará un controlador de excepciones para un error específico, es decir, un error de concurrencia, que se puede examinar mediante el objeto [DbConcurrencyException](#). Aquí este error se controlará mediante la presentación de información para el usuario.

## Para agregar un control específico para el error DbConcurrencyException

1. Si la aplicación sigue ejecutándose, salga del modo de ejecución para regresar al Editor de código.
2. Agregue una segunda instrucción **catch** sobre la existente en el método **updateDatabase**.
3. Pase el objeto **DbConcurrencyException** al procedimiento **createMessage**, que creará en la sección siguiente.

```
// Visual J#
private void updateDatabase()
{
    try
    {
        // Update the database with the changes from dsAuthors1.
        sqlDataAdapter1.Update(dsAuthors1.GetChanges());
        dsAuthors1.AcceptChanges();
        MessageBox.Show("The update was successful!");
    }
    catch (DbConcurrencyException dbcx)
    {
        createMessage(dbcx);
    }
    catch (System.Exception ex)
```

```

    {
        MessageBox.Show(ex.get_Message(), ex.GetType().ToString());
    }
}

```

## Presentar opciones al usuario

El código que acaba de escribir llama al procedimiento **createMessage** para mostrar información de error al usuario. En este tutorial, usará un cuadro de mensaje para mostrar al usuario las distintas versiones del registro y permitirle elegir si desea sobrescribir el registro con los nuevos cambios o cancelar la edición.

**Nota** En aras de la simplicidad, este tutorial usa el segundo conjunto de datos (**DsAuthors2**) como origen de datos para recuperar el registro actual de la base de datos. En una aplicación real se requeriría el origen de datos propiamente dicho para recuperar el valor actual del registro que provocó el error.

### Para crear el procedimiento createMessage

- Cree el controlador de errores; para ello, agregue el código siguiente en el Editor de código:

```

// Visual J#
private void createMessage(DBConcurrencyException dbcx)
{
    // Declare variables to hold the row versions for display
    // in the message box.
    System.String strInDs = "Original record in dsAuthors1:\n";
    System.String strInDB = "Current record in database:\n";
    System.String strProposed = "Proposed change:\n";
    System.String strPromptText = "Do you want to overwrite the current " +
        "record in the database with the proposed change?\n";
    System.String strMessage;
    System.Windows.Forms.DialogResult response;

    // Loop through the column values.
    DataRow rowInDB = dsAuthors2.get_authors().FindByau_id(dbcx.get_Row().get_Item("Au_ID")
).ToString());
    for (int i = 0; i < dbcx.get_Row().get_ItemArray().length; i++)
    {
        strInDs = strInDs + dbcx.get_Row().get_Item(i, DataRowVersion.Original) + "\n";
        strInDB = strInDB + rowInDB.get_Item(i, DataRowVersion.Current) + "\n";
        strProposed = strProposed + dbcx.get_Row().get_Item(i, DataRowVersion.Current) + "\n";
    }

    // Create the message box text string.
    strMessage = strInDs + "\n" + strInDB + "\n" + strProposed + "\n" + strPromptText;
    // Display the message box.
    response = MessageBox.Show(strMessage, dbcx.get_Message(), MessageBoxButtons.YesNo);
    processResponse(response);
}

```

## Procesar la respuesta del usuario

También necesitará código para procesar la respuesta del usuario al cuadro de mensaje. Las opciones son sobrescribir o no sobrescribir el registro actual de la base de datos con el cambio propuesto. Si el usuario responde afirmativamente, se llama al método [Merge](#) de **DsAuthors1** con el argumento *preserveChanges* establecido en **true**. Así, se extraen las versiones originales de las filas de datos de **DsAuthors2** y se combinan con las versiones actuales de las filas de datos de **DsAuthors1**. Esto hará que el intento de actualización sea satisfactorio, ya que ahora la versión original del registro coincide con el de la base de datos.

### Para procesar la respuesta del usuario en el cuadro de mensaje

- Agregue el código siguiente al Editor de código:

```
// Visual J#
private void processResponse(System.Windows.Forms.DialogResult response)
{
    // Execute the appropriate code depending on the button selected
    // in the message box.
    if(response == System.Windows.Forms.DialogResult.Yes)
    {
        dsAuthors1.Merge(dsAuthors2, true);
        sqlDataAdapter1.Update(dsAuthors1);
        dsAuthors1.AcceptChanges();
    }
    else if (response == System.Windows.Forms.DialogResult.No)
    {
        dsAuthors1.Merge(dsAuthors2);
    }
}
```

## Restablecer los datos

Para restablecer el formulario, se modificará el método **Form1\_Load** para que se ejecute cuando se haga clic en el botón **Reset**.

**Nota** Un solo método puede ser el controlador de eventos de múltiples eventos de múltiples objetos. Para obtener más información, vea [Conectar múltiples eventos a un solo controlador de eventos en formularios Windows Forms](#).

### Para crear un controlador de eventos para el botón btnReset

1. Abra el formulario en el diseñador.
2. Haga clic en el botón **Restablecer**.
3. En la ventana **Propiedades**, haga clic en el botón **Eventos** de la barra de herramientas.
4. Busque el evento **Click** y, a continuación, haga clic en la flecha para ver todos los métodos que podrían responder al evento.

En este caso, aparece **Form1\_Load** porque tiene la firma correcta para un evento **button.click**.

5. Seleccione **Form1\_Load**.

Se genera automáticamente el código que asocia el método **Form1\_Load** al evento **btnReset.Click**. **Form1\_Load** ahora responde a **Form1.Load** y **btnReset.Click**.

**Sugerencia** Para ver el código generado, haga doble clic en el formulario y expanda la sección "Código generado por el Diseñador de Windows Forms", que aparece atenuada.

## Ejecutar la aplicación

1. Presione F5 para ejecutar la aplicación.
2. Haga clic en el botón **Actualizar**.

Se producirá el error de concurrencia y aparecerá el cuadro de mensaje de error.

### Vea también

[Agregar nuevas conexiones de datos en el Explorador de servidores](#) |

[Actualizar la base de datos con un adaptador de datos y el conjunto de datos](#) |

[Actualizaciones de conjuntos de datos en Visual Studio .NET](#) | [Conjuntos de datos ADO.NET](#) | [Control de concurrencia en ADO.NET](#)

| [Depurar código administrado](#) | [Implementar aplicaciones](#) | [Tutoriales sobre datos](#)

# Tutorial: asignar tablas del origen de datos a tablas de conjunto de datos con Visual J#

La estructura de tablas en un conjunto de datos no siempre coincide con la estructura de tablas de un origen de datos. Pueden existir distintas razones para ello:

- Si el conjunto de datos se creó a partir de un esquema existente que utilice nombres diferentes de los del origen de datos.
- Si desea cambiar los nombres de los elementos del conjunto de datos por razones de comodidad, legibilidad, traducción hacia o desde un idioma diferente, o cualquier otra razón.
- Si desea controlar los nombres de los miembros de datos con tipo al generar un conjunto de datos a partir del adaptador.

Para hacer coincidir las tablas y columnas del origen de datos con las del conjunto de datos, puede establecer la propiedad **TableMappings** de un adaptador de datos. Esta propiedad contiene información que establece una correspondencia entre las columnas de la tabla del origen de datos y las columnas de la tabla del conjunto de datos. Cuando el adaptador lee datos del origen de datos, se usa la propiedad **TableMappings** para determinar en qué lugar del conjunto de datos se van a escribir los datos.

Este tutorial le ofrece una breve información general sobre cómo configurar las asignaciones de tablas en un adaptador de datos. En el escenario del tutorial se da por supuesto que está trabajando con un esquema que le ha proporcionado otra persona. Asignará las columnas de una tabla de la base de datos (en este caso, la tabla **Customers** de la base de datos Northwind) a las columnas definidas en el esquema.

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Northwind de SQL Server.

El tutorial está dividido en varias partes más pequeñas:

- Crear un formulario Windows Forms. El formulario sólo se utiliza para indicarle cómo crear la asignación; podría utilizar cualquier formulario o componente.
- Crear un conjunto de datos con un esquema predefinido. El tutorial contiene un esquema sencillo, pero completo, que puede copiar y pegar en Visual Studio para simular un esquema que podría obtener de otra persona.
- Crear y configurar el adaptador de datos en el que asignará el origen de datos al conjunto de datos.
- Agregar el control **DataGrid** para mostrar el efecto de la asignación.

## Crear el proyecto y el formulario

Este tutorial utiliza un formulario Windows Forms como contenedor en el que creará el adaptador de datos y el conjunto de datos. No es necesario que utilice un formulario Windows Forms, también podría utilizar una página de formularios Web Forms o un componente, pero un formulario Windows Forms facilita el trabajo y le permite utilizar un control **DataGrid** posteriormente para ver el efecto de las asignaciones.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, seleccione **Proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#** y, en el panel **Plantillas**, elija **Aplicación para Windows**.
3. Asigne un nombre al proyecto que sea único y cumpla las convenciones de nomenclatura utilizadas. Por ejemplo, podría denominar este proyecto como **Walkthrough\_Mappings**. Después de asignar un nombre, haga clic en **Aceptar**.

Visual Studio crea un proyecto nuevo y muestra un formulario nuevo en el Diseñador de Windows Forms.

## Crear el esquema y el conjunto de datos

Para el tutorial, imagine que alguien le ha proporcionado un esquema que define una tabla Customers. El esquema define una tabla **Customers** que es ligeramente distinta de la que tiene en la base de datos Northwind.

En esta sección, agregará un esquema al proyecto. El diseño del esquema real (el XML que define el esquema) se proporciona a continuación. De forma predeterminada, el Diseñador XML genera un archivo de clase basándose en el esquema y derivado de la clase [DataSet](#). El archivo de clase de resultado define la clase del conjunto de datos con tipo que utiliza en la aplicación.

Para obtener más información acerca de los conjuntos de datos y del modo en que se definen, vea [Introducción a los conjuntos de datos](#).

## Para agregar el esquema

1. En el **Explorador de soluciones**, haga clic con el botón secundario en el nombre del proyecto, elija **Agregar** y, a continuación, elija **Agregar nuevo elemento**.
2. En el panel **Plantillas**, elija **Conjunto de datos**, llame al nuevo elemento **dsCustomers** y, a continuación, haga clic en **Abrir**.

El Diseñador XML se abre con un nuevo esquema del conjunto de datos en blanco.

3. Haga clic en la ficha **XML** de la parte inferior.
4. Copie el texto del esquema siguiente y, a continuación, péguelo en la vista XML del diseñador, sobrescribiendo completamente el esquema que ya existe en el archivo.

**Sugerencia** Si se reemplazan los símbolos < y > por &lt; y &gt; cuando pegue, elimine el texto pegado y, a continuación, vuelva a pegar el esquema haciendo clic con el botón secundario del *mouse* en la superficie de diseño y seleccionando **Pegar como HTML** del menú contextual.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="dsCustomers" targetNamespace="http://www.tempuri.org/dsCustomers.xsd" xmlns:
s="http://www.tempuri.org/dsCustomers.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xm
l:msdata="urn:schemas-microsoft-com:xml-msdata" elementFormDefault="qualified">
  <xs:element name="dsCustomers" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CustID" type="xs:string" />
              <xs:element name="CompName" type="xs:string" />
              <xs:element name="ContName" type="xs:string" minOccurs="0" />
              <xs:element name="ContTitle" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    <xs:key name="Constraint1" msdata:PrimaryKey="true">
      <xs:selector xpath="." />
      <xs:field xpath="CustID" />
    </xs:key>
  </xs:element>
</xs:schema>
```

5. Guarde el esquema y ciérrelo.
6. Compruebe que tiene un archivo de clase de conjunto de datos basado en el esquema. En Explorador de soluciones, haga clic en el botón **Mostrar todos los archivos** y, a continuación, expanda el nodo del nuevo esquema. Debería ver un archivo de clase (con la extensión .jsl) debajo del esquema.

Ahora que tiene un esquema y un archivo de clase de conjunto de datos relacionados, puede agregar una instancia del conjunto de datos al formulario.

## Para agregar el conjunto de datos al formulario

1. Haga clic en la ficha **Form1** de la parte superior para volver al Diseñador de Windows Forms y al formulario.
2. En la ficha **Datos** del Cuadro de herramientas, arrastre el elemento **DataSet** hasta el formulario.

Se mostrará el cuadro de diálogo **Agregar conjunto de datos**.

3. Seleccione **Conjunto de datos con tipo** y, a continuación, en la lista **Nombre**, elija el conjunto de datos **Walkthrough\_Mappings.dsCustomers**. Al nombre del conjunto de datos se le agrega el prefijo del nombre de proyecto.

Al formulario situado debajo del nombre dsCustomers1 se le agrega una instancia del conjunto de datos.

## Crear un adaptador de datos y una asignación de tablas

Con el conjunto de datos definido, puede configurar la otra mitad del procedimiento del acceso a datos: el adaptador de datos que leerá datos de la base de datos y lo rellenará en el conjunto de datos. El adaptador de datos contendrá la asignación de tabla donde coinciden las columnas de la base de datos y las del conjunto de datos.

### Agregar el adaptador de datos

El primer paso es crear el adaptador de datos, lo que puede hacer mediante un asistente.

#### Para agregar un adaptador de datos

1. En la ficha **Datos** del Cuadro de herramientas, arrastre el elemento **OleDbDataAdapter** hasta el formulario.

Se abrirá el cuadro de diálogo **Asistente para la configuración del adaptador de datos**.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Northwind de SQL Server.
  - b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
  - c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle
FROM Customers
```

Para obtener ayuda para generar la instrucción SQL, haga clic en **Generador de consultas** para iniciar el **Generador de consultas**.

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Customers. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

Cuando el asistente haya finalizado, tendrá una conexión (**OleDbConnection1**) que contiene información sobre cómo tener acceso a la base de datos. También tendrá un adaptador de datos (**OleDbDataAdapter1**) que contiene una consulta que define la tabla y las columnas de la base de datos a la que desea tener acceso.

### Crear la asignación de tablas

Ahora que ya tiene un adaptador de datos en el formulario, puede configurar sus asignaciones de tablas.

#### Para crear las asignaciones

1. Seleccione el adaptador de datos de la parte inferior del formulario.
2. En la ventana Propiedades, haga clic en el botón de la propiedad **TableMappings**.

Se mostrará el cuadro de diálogo **Asignaciones de tabla**.

3. Marque **Utilizar un conjunto de datos para sugerir nombres de tablas y columnas** y en la lista desplegable, seleccione **Walkthrough\_Mappings.dsCustomers**. (Si le ha dado otro nombre al proyecto, el prefijo de la lista será distinto.)
4. En la lista **Tabla de conjunto de datos**, seleccione **Customers**.
5. Cambie los nombres de columnas de la lista **Columnas de conjuntos de datos** individualmente utilizando la lista desplegable que está disponible para cada columna. Cree las asignaciones siguientes:

Columnas de origen	Columnas del conjunto de datos
CustomerID	CustID
CompanyName	CompName
ContactName	ContName
ContactTitle	ContTitle

Cuando haya terminado, el cuadro de diálogo **Asignaciones de tabla** presentará la apariencia siguiente:

Asignaciones de tablas

Especificar, para cada columna de la tabla de origen, el nombre de la columna correspondiente en el conjunto de datos.

☒ Utilizar un conjunto de datos para sugerir nombres de tabla y de columna.

Conjunto de datos:  
WindowsApplication1.dsCustomers

Tabla de origen: Table      Tabla de conjuntos de datos: Customers

Asignaciones de columnas:

Columnas de origen	Columnas de conjuntos de dato
CustomerID	CustID
CompanyName	CompName
ContactName	ContName
ContactTitle	ContTitle

Eliminar      Restablecer

6. Haga clic en Aceptar para cerrar el cuadro de diálogo **Asignaciones de tablas**.

## Utilizar controles para mostrar nombres asignados

En el paso final del tutorial, agregue un control **DataGrid** que muestra cómo se utilizan las asignaciones en el conjunto de datos. Cuando enlace la cuadrícula a la tabla del conjunto de datos, verá que la tabla del conjunto de datos contiene los nombres asignados.

### Para agregar y enlazar un control DataGrid

1. Agregue un control **DataGrid** de la ficha **Windows Forms** del **Cuadro de herramientas**.
2. Seleccione el control **DataGrid** y abra la ventana **Propiedades**.
3. Establezca las siguientes propiedades de enlace de datos.
  - Establezca la propiedad **DataSource** en **dsCustomers1**.
  - Establezca la propiedad **DataMember** en **Customers**.

Los encabezados de columna coincidirán con los del esquema.

Para poder ejecutar el formulario y ver los datos asignados que se utilizan, necesitará llenar el conjunto de datos.

### Para llenar el conjunto de datos

- Haga doble clic en el formulario (no en el control) para crear un controlador de eventos para el evento **Load** del formulario y agréguele la línea siguiente:

```
// Visual J#
oleDbDataAdapter1.Fill(dsCustomers1);
oleDbConnection1.Close();
```

Por último, puede comprobar el formulario para ver la asignación en funcionamiento.

### Para comprobar el formulario

1. Presione F5 para ejecutar el formulario.
2. Cuando se muestre el formulario, confirme que los datos de la tabla Customers de Northwind se muestran en la cuadrícula utilizando los nombres de columna asignados.

## Pasos siguientes

En este tutorial se ha proporcionado una breve introducción a la asignación de tablas en adaptadores de datos. La asignación se puede utilizar no sólo para cambiar nombres de columnas, sino con fines más sofisticados, entre ellos:

- Asignar columnas de una tabla a otra columna completamente distinta del conjunto de datos.

- Utilizar la asignación para limitar el número de columnas creadas en un conjunto de datos. Esto es útil si el adaptador de datos lee todas las columnas de la tabla de la base de datos, pero sólo necesita unas pocas en una tabla del conjunto de datos.
- Asignar una expresión de columna de un conjunto de datos a una columna de la base de datos. Esto le permite calcular valores del conjunto de datos y escribirlos en la base de datos.
- Provocar errores bajo determinadas circunstancias, incluidas las no coincidencias.

Para obtener más información, vea [Asignaciones de tablas en adaptadores de datos](#).

### **Vea también**

[Introducción a los adaptadores de datos](#) | [Introducción a conjuntos de datos](#) | [Introducción a los esquemas XML](#) | [Crear conjuntos de datos y esquemas XML](#) | [Crear un esquema XML con el Diseñador XML](#) | [Tutoriales sobre datos](#)



# Tutorial: crear un archivo XML con un esquema XML asociado con Visual J#

Este tutorial muestra las características del trabajo con archivos XML que tienen esquemas asociados. Le guiará a través del proceso de creación de un archivo XML y un esquema XML basados en el lenguaje de definición de esquemas XML (XSD). A continuación, creará una asociación entre el esquema XML y el archivo XML. Una vez establecida la asociación, trabajará en el archivo XML, en la vista XML, para ver las características avanzadas del editor XML. En este tutorial, el archivo XML representa una lista de compañías. Verá cómo el archivo XML contiene los datos mientras que el esquema XML sólo define esos datos.

**Nota** Mediante la asociación de un esquema XML con el archivo XML, se habilita la finalización de instrucciones en el editor XML.

## Crear un nuevo proyecto de aplicación para Windows

Para comenzar el tutorial, deberá crear primero una aplicación para Windows.

### Para crear un proyecto nuevo y denominarlo "XMLCustomerList"

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows**.
3. Asigne al proyecto el nombre **XMLCustomerList** y, a continuación, haga clic en **Aceptar** para crear el proyecto.

Visual Studio agregará el proyecto XMLCustomerList al Explorador de soluciones.

## Agregar un nuevo elemento Archivo XML al proyecto

Dado que este tutorial requiere un archivo XML, necesita agregar uno al proyecto.

### Para agregar un nuevo elemento Archivo XML al proyecto

1. En el menú **Proyecto**, elija **Agregar nuevo elemento**.  
Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.
2. En el área Plantillas del cuadro de diálogo **Agregar nuevo elemento**, seleccione **Archivo XML**.
3. Dele al archivo XML el nombre **CustomerList** y, a continuación, haga clic en **Abrir**.

Visual Studio agregará un archivo denominado **CustomerList.xml** al proyecto y lo cargará automáticamente en el diseñador.

## Agregar un nuevo elemento Esquema XML al proyecto

Dado que este tutorial requiere un esquema XML, necesita agregar uno al proyecto.

### Para agregar un nuevo esquema XML al proyecto

1. En el menú **Proyecto**, elija **Agregar nuevo elemento**.  
Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.
2. En el área Plantillas del cuadro de diálogo **Agregar nuevo elemento**, seleccione **Esquema XML**.
3. Asigne al esquema el nombre **CustomerListSchema** y, a continuación, haga clic en **Abrir**.

Visual Studio agregará un archivo denominado **CustomerListSchema.xsd** al proyecto y lo cargará automáticamente en el diseñador.

## Agregar una definición de tipo sencilla al esquema

Ahora deberá definir los elementos que contendrán los datos en el archivo XML. Dado que un esquema XML define los datos en un archivo XML asociado, cree las definiciones del elemento en el esquema.

La primera definición será de un elemento de tipo simple que se utilizará para definir un código postal estándar de Estados Unidos. En este tutorial sólo utilizaremos códigos de 5 dígitos.

### Para crear un elemento de tipo simple que represente un código postal de 5 dígitos

1. En la ficha **Esquema XML** del Cuadro de herramientas, arrastre un objeto **simpleType** hasta la superficie del diseñador.
2. Seleccione el nombre predeterminado **simpleType1** y cambie el nombre de este tipo a **postalCode**.
3. Utilice la tecla TABULADOR para ir una celda a la derecha y seleccionar **positiveInteger** en la lista desplegable.
4. Utilice la tecla TABULADOR para ir a la fila siguiente.
5. Haga clic en el cuadro desplegable.

La única opción es aspecto. Esto se debe a que los tipos simples no pueden incluir elementos o atributos como parte de los modelos de contenido. Sólo se pueden utilizar aspectos para generar tipos simples.

6. Utilice la tecla TABULADOR para ir una celda a la derecha y seleccionar **pattern** en la lista desplegable.
7. Utilice la tecla TABULADOR para ir de nuevo una celda a la derecha y escriba **\d{5}**.

El aspecto de modelo le permite proporcionar expresiones regulares. La expresión regular **\d{5}** indica que el contenido del tipo **postalCode** está restringido a 5 dígitos. Las expresiones regulares no entran dentro del ámbito de este tutorial, pero se puede ver cómo utilizar el aspecto de modelo junto con el tipo de datos elegido para permitir únicamente datos específicos en un tipo simple.

Si cambia el esquema a una vista XML, debería ver el código siguiente en las etiquetas del esquema del nivel raíz (lo que significa que el ejemplo de código no incluye la parte de declaración real del esquema, ni tampoco incluye las etiquetas reales del esquema que se denominan etiquetas del nivel documento o raíz):

```
<xs:simpleType name="postalCode">
  <xs:restriction base="xs:positiveInteger">
    <xs:pattern value="\d{5}" />
  </xs:restriction>
</xs:simpleType>
```

8. En el menú **Archivo**, seleccione **Guardar todo**.

## Agregar una definición de tipo compleja al esquema

La siguiente definición será un elemento de tipo complejo que se utilizará para definir una dirección. Como parte de esta definición de tipo complejo, se utilizará el tipo simple creado en los pasos anteriores.

### Para crear un elemento de tipo complejo que represente una dirección postal estándar de Estados Unidos

1. Cambie de nuevo a la vista **Esquema**.
2. En la ficha **Esquema XML** del Cuadro de herramientas, arrastre un objeto **complexType** hasta la superficie del diseñador.
3. Seleccione el nombre predeterminado **complexType1** y cambie el nombre de este tipo a **usAddress**. No seleccione un tipo de datos para este elemento.
4. Mediante la tecla TABULADOR, vaya a la fila siguiente.
5. Haga clic en el cuadro de lista desplegable para ver las diversas opciones de elementos que puede agregar a un tipo complejo. Puede seleccionar un elemento, pero en el resto del tutorial sólo presionará la tecla TABULADOR en esta celda porque se trata del elemento predeterminado.
6. Mediante la tecla TABULADOR, vaya una celda a la derecha y escriba **Name**.
7. Utilice la tecla TABULADOR para ir una celda a la derecha y establezca el tipo de datos en **string**.
8. Repita los pasos que van del 4 al 7 y cree nuevas filas en el elemento **usAddress** para lo siguiente:

Nombre del elemento	Tipo de datos
<b>Street</b>	<b>string</b>
<b>City</b>	<b>string</b>
<b>State</b>	<b>string</b>
<b>Zip</b>	<b>postalCode</b>

Observe el tipo de datos que está asignado al elemento **Zip**. Es el tipo simple **postalCode** que ha creado anteriormente.

Si cambia a la vista XML, debería ver el código siguiente en las etiquetas del esquema del nivel raíz (lo que significa que el ejemplo de código no incluye la parte de declaración real del esquema, ni tampoco incluye las etiquetas reales del esquema que se denominan etiquetas del nivel documento o raíz):

```
<xs:simpleType name="postalCode">
```

```

        <xs:restriction base="xs:positiveInteger">
            <xs:pattern value="\d{5}" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="usAddress">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" />
            <xs:element name="Street" type="xs:string" />
            <xs:element name="City" type="xs:string" />
            <xs:element name="State" type="xs:string" />
            <xs:element name="Zip" type="postalCode" />
        </xs:sequence>
    </xs:complexType>

```

Ahora ha definido dos tipos independientes que se pueden utilizar en definiciones de elementos y en tipos.

9. En el menú **Archivo**, seleccione **Guardar todo**.

## Agregar los elementos principales al esquema

Ahora que ha definido algunos tipos de datos, cree la definición de datos real para el archivo XML que se va a crear. El archivo XML contendrá los datos de la lista de clientes, para eso cree el elemento real que definirá los datos que serán válidos en el archivo XML.

### Para crear el elemento Customer

1. Cambie a la vista **Esquema**.
2. En la ficha **Esquema XML** del Cuadro de herramientas, arrastre un objeto **element** hasta la superficie del diseñador.
3. Seleccione el nombre predeterminado **element1** y cambie su nombre por **customer**. No seleccione un tipo de datos para este elemento.
4. Utilizando la tecla TABULADOR, vaya a la celda central de la fila siguiente y escriba **CompanyName**.
5. Con la tecla TABULADOR, vaya a una celda a la derecha y establezca el tipo de datos en **string**.
6. Repita los pasos 4 y 5 y cree nuevas filas en el elemento **Customer** para lo siguiente:

Nombre del elemento	Tipo de datos
<b>ContactName</b>	<b>string</b>
<b>Email</b>	<b>string</b>
<b>Phone</b>	<b>string</b>
<b>BillToAddress</b>	<b>usAddress</b>
<b>ShipToAddress</b>	<b>usAddress</b>

Observe el tipo de datos que está asignado a los elementos **BillToAddress** y **ShipToAddress**. Es el tipo complejo **usAddress** creado anteriormente. Podría haber definido tipos simples para los elementos **Email**, **Phone**, etc.

Si cambia el esquema a una vista XML, debería ver el código siguiente en las etiquetas del esquema del nivel raíz (lo que significa que el ejemplo de código no incluye la parte de declaración real del esquema, ni tampoco incluye las etiquetas reales del esquema que se denominan etiquetas del nivel documento o raíz):

```

<xs:simpleType name="postalCode">
    <xs:restriction base="xs:positiveInteger">
        <xs:pattern value="\d{5}" />
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="usAddress">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="City" type="xs:string" />
        <xs:element name="State" type="xs:string" />
        <xs:element name="Zip" type="postalCode" />
    </xs:sequence>

```

```

</xs:complexType>
<xs:element name="Customer">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CompanyName" type="xs:string" />
      <xs:element name="ContactName" type="xs:string" />
      <xs:element name="Email" type="xs:string" />
      <xs:element name="Phone" type="xs:string" />
      <xs:element name="ShipToAddress" type="usAddress" />
      <xs:element name="BillToAddress" type="usAddress" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

7. En el menú **Archivo**, seleccione **Guardar todo**.

Para crear múltiples instancias de los datos del cliente dentro del documento XML, crearemos un elemento denominado **customerList** que contendrá todos los elementos **customer** individuales.

#### Para crear el elemento customerList

1. En la ficha **Esquema XML** del Cuadro de herramientas, arrastre un elemento hasta la superficie del diseñador.
2. Seleccione el nombre predeterminado **element1** y cambie su nombre por **customerList**. No seleccione un tipo de datos para este elemento.
3. Seleccione el elemento **customer** (creado previamente) y arrástrelo al elemento **customerList**.

Las cuadrículas de diseño individuales se enlazan para representar la estructura jerárquica de los datos.

4. En el menú **Archivo**, seleccione **Guardar todo**.

### Asociar el esquema y el archivo XML

En esta parte del tutorial, empezará a trabajar con el archivo XML. En el archivo XML, agregará una referencia al esquema que acaba de crear.

#### Para crear una asociación entre el archivo XML y el esquema XML

1. En el Explorador de soluciones, haga doble clic en el archivo CustomerList.xml.
2. El archivo XML se abre en el diseñador en la vista XML.
3. En la ventana **Propiedades**, haga clic en la celda que se encuentra a la derecha de la propiedad **targetSchema** y seleccione **http://tempuri.org/CustomerListSchema.xsd**.

Visual Studio agrega una referencia al esquema del archivo CustomerList.xml y agrega las etiquetas **<customerList>**.

### Agregar datos al archivo XML

Ahora el archivo XML está preparado para recibir datos. Mediante la asociación de un esquema al archivo XML, el editor XML ahora está 'enterado' de los elementos válidos que se pueden incluir en el archivo XML, como se demostrará en esta sección.

#### Para agregar datos al archivo customerList.xml

1. En la vista **XML** del archivo **customerList.xml**, sitúe el cursor entre las etiquetas de apertura **<customerList>** y de cierre **</customerList>**.
2. Escriba **<**. Escriba **c** para seleccionar el elemento **customer**.
3. Escriba **>** para cerrar la etiqueta.
4. Escriba **<** y se mostrará una lista de elementos válidos.

Mediante la asociación de un esquema XML con el archivo XML, se habilita la finalización de instrucciones en el editor XML.

### Pasos siguientes

Algunos de los siguientes pasos habituales incluyen:

- Agregar algunos datos en la vista XML y, a continuación, cambiar a la vista Datos. Explorar las diferentes formas de agregar

datos a los archivos XML mediante el Diseñador XML.

- Después de agregar algunos datos, validar el documento.
- Generar un conjunto de datos desde el esquema.

### **Vea también**

[Datos y esquemas XML](#) | [Archivos XML](#) | [Crear nuevos archivos XML](#) | [Asignar esquemas a archivos XML](#) | [Comprobar la corrección y validez de XML](#) | [Elementos, atributos y tipos XML](#) | [Crear objetos DataSet con tipo de ADO.NET a partir de esquemas](#) [Tutoriales sobre datos](#)

# Tutorial: crear un esquema XML con el Diseñador XML en Visual J#

En este tutorial creará un esquema de pedido XML como parte de un proyecto de aplicación para Windows. El tutorial consta de tres secciones principales:

1. Crear un proyecto de aplicación para Windows y agregar un esquema XML.
2. Crear una tabla relacional:
  - a. Agregar y definir un objeto **simpleType** nuevo.
  - b. Agregar y definir un objeto **complexType** nuevo.
  - c. Agregar y definir un objeto **Element** nuevo.
3. Modificar el XML generado mediante el editor XML.

## Crear un proyecto de aplicación para Windows y agregar un esquema XML

### Para crear un proyecto de Aplicación para Windows nuevo

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows**.
3. Asigne al proyecto el nombre **SampleSchema**.

### Para agregar un esquema XML al proyecto

- En el menú **Proyecto**, seleccione **Agregar nuevo elemento** y, a continuación, haga doble clic en el icono **Esquema XML** en el cuadro de diálogo **Agregar nuevo elemento**.

Aparecerá el Diseñador XML.

## Definir los tipos simple y complejo

Antes de crear la tabla relacional, primero creará definiciones de tipos simple y complejo que utilizará para dar formato a elementos específicos del esquema de pedido. Los nuevos tipos se crean utilizando tipos de datos XML existentes, como string e integer.




En primer lugar definirá un tipo simple, que se denominará **stateCode**. Este tipo simple se utilizará para limitar el tamaño de una cadena a dos caracteres.

### Para agregar un objeto simpleType XML al proyecto

1. Si aún no está abierto, haga doble clic en el archivo **XMLSchema1.xsd** para abrir el Diseñador XML.
2. Haga clic en la ficha **Esquema XML** del Cuadro de herramientas y arrastre un objeto **simpleType** hasta la superficie del diseñador.
3. Cambie el nombre del objeto **simpleType** haciendo clic en el primer cuadro de texto del encabezado y reemplazando **simpleType1** por **stateCode**.
4. Establezca el tipo base del tipo **stateCode** haciendo clic en la lista desplegable del encabezado y seleccionando **string**.
5. Presione la tecla TABULADOR para desplazarse hasta la primera celda de la fila siguiente.
6. Seleccione **facet** en la lista desplegable.
7. Presione la tecla TABULADOR para ir a la celda siguiente y seleccione **length** en la lista desplegable.
8. Presione la tecla TABULADOR para ir a la tercera celda de la misma fila y escriba el valor **2**.

De esta manera se exige que el valor escrito en el campo **State** tenga dos caracteres.

StateCode debería tener esta apariencia en la vista Esquema:

	CódigoPostal	string
	length	2
		

9. Haga clic en la ficha **XML** en la parte inferior izquierda del **Diseñador XML**, para ver el código XML que se ha agregado:

```
<xs:simpleType name="stateCode">
```

```

<xs:restriction base="xs:string">
  <xs:length value="2" />
</xs:restriction>
</xs:simpleType>

```

Este tipo simple **stateCode** se utilizará para definir el elemento **State** del tipo complejo que creará en la siguiente sección.

El **complexType** denominado **addressType** define un conjunto de elementos que aparecerán en cualquier elemento con tipo como **addressType**. Por ejemplo, un elemento **billTo** incluirá información de nombres, direcciones y fechas cuando su tipo se establezca en el **addressType** definido anteriormente. Mediante la creación del tipo complejo y utilizándolo en un elemento, se genera una relación anidada. Para obtener más información, vea [Crear tipos XML complejos](#).

### Para agregar un objeto complexType XML al proyecto

1. Haga clic en la ficha **Esquema** del Diseñador XML.
2. Arrastre un objeto **complexType** desde la ficha **Esquema XML** del Cuadro de herramientas a la superficie de diseño.
3. Cambie **complexType1** a **addressType** para asignar nombre al tipo.
4. Agregue un atributo XML a **addressType** haciendo clic en la primera celda de la primera fila y seleccionando **element** en la lista desplegable.
5. En la segunda columna, cambie **element1** a **Name**.
6. En la tercera columna, acepte el valor predeterminado **string**.
7. Agregue los siguientes elementos XML y establezca los nombres y tipos de la forma siguiente:

Nombre del elemento	Tipo de datos
<b>Street</b>	<b>string</b>
<b>City</b>	<b>string</b>
<b>State</b>	<b>stateCode</b>
<b>PostalCode</b>	<b>integer</b>

**AddressType** debería tener esta apariencia en la vista Esquema:



Tipodirección		
E	Nombre	string
E	Calle	string
E	Ciudad	string
E	Provincia	string
E	Código Postal	integer

8. Para ver el código XML que se ha agregado al archivo .xsd, haga clic en la ficha XML en la parte inferior del diseñador. Verá el siguiente código XML nuevo:

```

<xs:complexType name="addressType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Street" type="xs:string"/>
    <xs:element name="City" type="xs:string"/>
    <xs:element name="State" type="stateCode"/>
    <xs:element name="PostalCode" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>

```

### Crear una tabla relacional

Cuando arrastra el objeto **element** del Cuadro de texto a la superficie de diseño, realmente agrega un elemento que contiene un **complexType** sin nombre. Al incluir el tipo complejo sin nombre se define el elemento para que sea una tabla relacional. A continuación, es posible agregar elementos adicionales bajo el **complexType** para definir los campos (o columnas) de relación. Si define uno de estos nuevos elementos como un nuevo **complexType** sin nombre, está creando una relación anidada dentro de la relación primaria con sus propias columnas únicas. Para obtener información detallada, vea [Tablas, columnas, claves y restricciones en esquemas XML](#).

Mediante la definición de nuevos elementos de tipo complejo sin nombre en el elemento **PurchaseOrder** o **Items** se crean anidamientos adicionales en el esquema. En un pedido puede haber muchos elementos **Item**, y en cada uno de ellos, muchos elementos adicionales (como precio, tamaño, etc.). En el procedimiento siguiente, se agrega un elemento **Items** a la tabla

relacional **purchaseOrder** y se califica como **complexType** sin nombre. Ya que está definiendo una nueva tabla relacional, esto hace que aparezca un nuevo elemento en la superficie de diseño. En la relación de nuevos Items, al agregar el elemento Item y establecer su tipo en **complexType** sin nombre, se crea otra tabla relacional, que también aparece en la superficie de diseño.

### Para agregar un elemento XML al proyecto

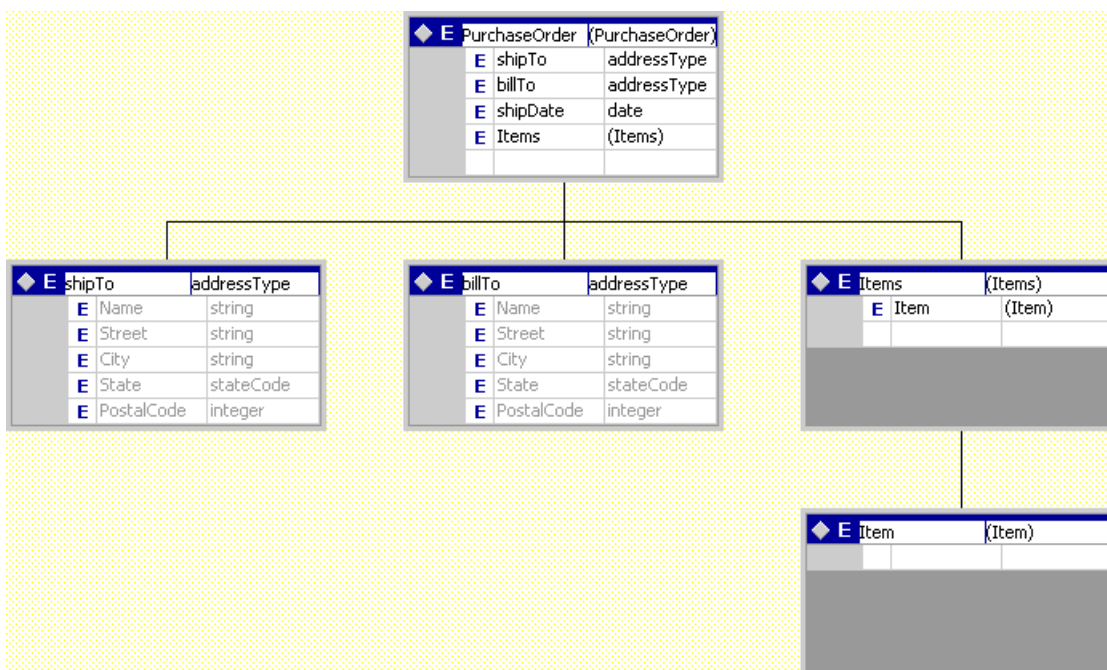
1. Haga clic en el Cuadro de herramientas y, en la ficha **Esquema XML**, arrastre un objeto **element** hasta la superficie de diseño.
2. Cambie **element1** a **PurchaseOrder** para asignar un nombre al elemento. Puede dejar el tipo de datos como (**PurchaseOrder**).
3. Agregue un elemento al pedido haciendo clic en la primera celda de la primera fila y seleccionando **element** en la lista desplegable.
4. Dele al elemento el nombre **shipTo** y establezca su tipo de datos en **addressType**.
5. Agregue los siguientes elementos XML y establezca los nombres y tipos de la forma siguiente:

Nombre del elemento	Tipo de datos
<b>billTo</b>	<b>addressType</b>
<b>shipDate</b>	<b>date</b>
<b>Items</b>	<b>complexType</b> sin nombre

Cuando escriba el elemento **Item** como anónimo, se agregará un elemento adicional a la superficie de diseño, que es otra tabla relacional.

6. En el **elemento Items**, agregue un elemento, asígnele el nombre **Item** y establezca su tipo en **ComplexType sin nombre**.

La orden de compra debería tener esta apariencia en la vista Esquema:



El código XML siguiente se ha agregado al archivo .xsd:

```

<xs:element name="PurchaseOrder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="shipTo" type="addressType"/>
      <xs:element name="billTo" type="addressType"/>
      <xs:element name="shipDate" type="xs:date"/>
      <xs:element name="Items">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Item">
              <xs:complexType>
                <xs:sequence />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
  
```



```
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

## Editar XML

Puede utilizar la ficha XML del Diseñador XML para modificar el código XML que se ha generado al agregar elementos y tipos a la superficie del diseñador. El editor XML incluye IntelliSense y la finalización de instrucciones. Las instrucciones no válidas se marcan con una línea roja ondulada. Al mover el *mouse* (ratón) sobre la instrucción incorrecta hace que aparezca un mensaje de error.

### Para editar XML

1. Haga clic en la ficha **XML** del Diseñador HTML para ver el código XML.
2. Dentro del elemento **Item**, cambie la etiqueta de cierre automático ( `<xs:sequence />` ) por etiquetas separadas de apertura y cierre ( `<xs:sequence></xs:sequence>` ). Si escribe `>` inmediatamente después de `sequence`, el Editor XML generará automáticamente la etiqueta de cierre.
3. Después de la etiqueta `<xs:sequence>` del elemento **Item**, escriba lo siguiente:

```
<xs:element name="Quantity" type="xs:integer"/>
<xs:element name="Price" type="xs:decimal"/>
<xs:element name="ProductID" type="xs:integer"/>
```

Ha creado tres nuevos elementos: **Quantity**, **Price** y **ProductID**, y ha definido los tipos de datos de cada uno de ellos.

4. A continuación, escriba `<invalid/>` y observe que la línea roja ondulada indica un error. Mueva el *mouse* sobre la línea roja ondulada para ver un mensaje de error. Los errores también aparecerán en la Lista de tareas.
5. Elimine la etiqueta `<invalid/>` para corregir el error.
6. Guarde el esquema.

El código XML situado debajo del elemento `Items` ahora debería parecerse al siguiente en la vista XML:

```
<xs:element name="Items">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Item">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Quantity" type="xs:integer"/>
            <xs:element name="Price" type="xs:decimal"/>
            <xs:element name="ProductID" type="xs:integer"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Vea también

[Introducción a los esquemas XML](#) | [Datos relacionales en esquemas XML](#) |

[Tablas, columnas, claves y restricciones en esquemas XML](#) | [Elementos, atributos y tipos XML](#) | [Referencia de esquemas XML \(XSD\)](#) | [Tutoriales sobre datos](#)

# Tutorial: leer datos XML en un conjunto de datos con Visual J#

ADO.NET proporciona métodos sencillos para trabajar con datos XML. En este tutorial, se creará una aplicación para Windows que cargará datos XML en un conjunto de datos. Después, el conjunto de datos se mostrará en un control **DataGrid**. Por último, se mostrará en un cuadro de texto un esquema XML basado en el contenido del archivo XML.

Este tutorial consta de cinco pasos principales:

1. Crear un proyecto de Visual J# nuevo.
2. Crear un archivo XML que será leído en el conjunto de datos.
3. Crear la interfaz de usuario.
4. Agregar código para crear el conjunto de datos, leer el archivo XML y mostrarlo en un control **DataGrid**.
5. Agregar código para mostrar en un control **TextBox** el esquema XML basado en el archivo XML.

## Crear un nuevo proyecto

En este paso, creará un proyecto de Visual J# que incluirá este tutorial.

### Para crear un nuevo proyecto

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto** para mostrar el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione **Proyectos de Visual J#** en el panel **Tipos de proyecto** y elija **Aplicación para Windows**.
3. Asigne al proyecto el nombre **ReadingXML** y haga clic en **Aceptar**.

Visual Studio agregará el proyecto **ReadingXML** al Explorador de soluciones y mostrará un nuevo formulario Windows Forms en el diseñador.

## Generar el archivo XML que será leído en el conjunto de datos

Debido a que este tutorial se centra en la lectura de datos XML en un conjunto de datos, se proporciona el contenido de un archivo XML. Este archivo XML debe encontrarse en una carpeta que sea accesible desde el equipo que va a utilizar para crear esta aplicación.

### Para crear el archivo XML que será leído en el conjunto de datos

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Archivo**.

Aparecerá el cuadro de diálogo **Nuevo archivo**.

2. Seleccione **Archivo XML** y haga clic en **Abrir**.

El archivo XML se carga en el diseñador y está listo para su edición.

3. Pegue el código siguiente en el editor, debajo de la declaración XML:

**Sugerencia** Haga clic con el botón secundario del *mouse* (ratón) y seleccione **Pegar como HTML**.

```
<Authors_Table>
  <authors>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <phone>408 496-7223</phone>
    <address>10932 Bigge Rd.</address>
    <city>Menlo Park</city>
    <state>CA</state>
    <zip>94025</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
```

```

    <au_fname>Margie</au_fname>
    <phone>415 986-7020</phone>
    <address>309 63rd St. #411</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94618</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
    <phone>415 548-7723</phone>
    <address>589 Darwin Ln.</address>
    <city>Berkeley</city>
    <state>CA</state>
    <zip>94705</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>267-41-2394</au_id>
    <au_lname>Hunter</au_lname>
    <au_fname>Anne</au_fname>
    <phone>408 286-2428</phone>
    <address>22 Cleveland Av. #14</address>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95128</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>274-80-9391</au_id>
    <au_lname>Straight</au_lname>
    <au_fname>Dean</au_fname>
    <phone>415 834-2919</phone>
    <address>5420 College Av.</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94609</zip>
    <contract>true</contract>
  </authors>
</Authors_Table>

```

4. En el menú **Archivo**, seleccione **Guardar XMLFile1 como**.

Aparecerá el cuadro de diálogo **Guardar archivo como**.

5. Llame al archivo **authors.xml** y haga clic en **Guardar**.

**Nota** Asegúrese de incluir la ruta de acceso completa, ya que la necesitará más adelante en el tutorial.

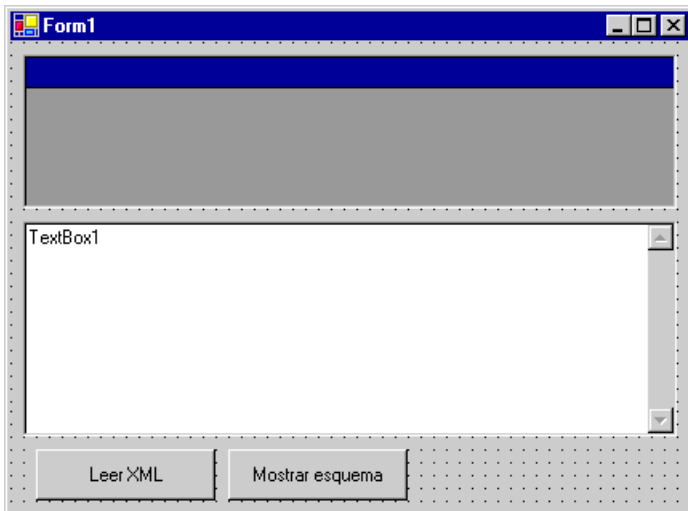
## Crear la interfaz de usuario

La interfaz de usuario para esta aplicación constará de los siguientes elementos:

- Un control **DataGrid** que mostrará el contenido del archivo XML como datos.
- Un control **TextBox** que mostrará el esquema XML para el archivo XML.
- Dos controles **Button**, como sigue:
  - Un botón lee el archivo XML en el conjunto de datos y lo muestra en el control **DataGrid**.
  - El segundo botón extrae el esquema del conjunto de datos y, a través de [StringWriter](#), lo muestra en el control

## TextBox.

### Diseño inicial de la aplicación



### Para agregar controles al formulario

1. En la ficha **Windows Forms** del Cuadro de herramientas, arrastre los controles siguientes hasta el formulario:
  - 1 control [DataGrid](#)
  - 1 control [TextBox](#)
  - 2 controles [Button](#)
2. Establezca las siguientes propiedades:

Control	Propiedad	Valor
TextBox1	Multiline	true
	ScrollBars	Vertical
Button1	Name	btnReadXML
	Text	Read XML
Button2	Name	btnShowSchema
	Text	Show Schema

### Crear el conjunto de datos que recibirá los datos XML

Deberá crear mediante programación un nuevo conjunto de datos denominado **authors**. Se declarará en el nivel de formulario para que sea accesible desde los controladores de eventos **button\_click**. Para obtener más información acerca de los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

### Para agregar el código para crear un nuevo conjunto de datos que recibirá los datos XML

1. Con **Form1.jsl** seleccionado en el Explorador de soluciones, haga clic en el botón **Ver código**.
2. En el área de declaraciones, declare un nuevo conjunto de datos denominado "authors":

```
// Visual J#  
DataSet dsAuthors = new DataSet("authors");
```

### Crear el controlador de eventos para leer los datos XML en el conjunto de datos

El botón **Leer XML** lee el archivo XML en el conjunto de datos y establece propiedades en el control **DataGrid** que lo enlazan al conjunto de datos.

### Para agregar código al controlador de eventos btnReadXML\_Click

1. En el Explorador de soluciones, seleccione **Form1** y haga clic en el botón **Diseñador de vistas**.
2. Haga doble clic en el botón **Leer XML**.

El Editor de código se abre en el controlador de eventos **btnReadXML\_Click**.

3. Escriba el código siguiente en el controlador de eventos **btnReadXML\_Click**:

```
// Visual J#
System.String filePath = "Complete path where you saved the XML file";
dsAuthors.ReadXml(filePath);
dataGridView1.SetDataSource(dsAuthors);
dataGridView1.SetDataMember("authors");
dataGridView1.SetCaptionText(dataGridView1.GetDataMember());
```

4. En el código del controlador de eventos **btnReadXML\_Click**, modifique la entrada `filePath` = para corregir la ruta de acceso.

## Crear el controlador de eventos para mostrar el esquema en el control TextBox

El botón Mostrar esquema crea un objeto **StringWriter** que se llena con el esquema y se muestra en el control **TextBox**.

### Para agregar código al controlador de eventos **btnShowSchema\_Click**

1. En el Explorador de soluciones, seleccione **Form1.jsl** y haga clic en el botón **Diseñador de vistas**.
2. Haga doble clic en el botón **Mostrar esquema**.

El Editor de código se abre en el controlador de eventos **btnShowSchema\_Click**.

3. Escriba el código siguiente en el controlador de eventos **btnShowSchema\_Click**.

```
// Visual J#
System.IO.StringWriter swXML = new System.IO.StringWriter();
dsAuthors.WriteXmlSchema(swXML);
textBox1.SetText(swXML.ToString());
```

## Ejecutar la aplicación

1. Presione F5 para ejecutar la aplicación.
2. Haga clic en el botón **Leer XML**.

La cuadrícula de datos muestra el contenido del archivo XML.

3. Haga clic en el botón **Mostrar esquema**.

El cuadro de texto muestra el esquema XML del archivo XML.

## Resultado de ejecutar la aplicación

The screenshot shows a Windows application window titled "Form1". It contains two main components: a data grid and a text box. The data grid, titled "authors", displays the following data:

au_id	au_fname	au_lname	phone	address	city
172-32-1176	White	Johnson	408 496-7223	10932 Bigge	Menlo Park
213-46-8915	Green	Margie	415 986-7020	309 63rd St.	Oakland
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin L	Berkeley

Below the data grid is a text box containing the XML schema for the "authors" table. The schema is as follows:

```
<xsd:schema id="Authors_Table" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="Authors_Table" msdata:IsDataSet="true" msdata:Locale="es-ES">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="authors">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="au_id" type="xsd:string" minOccurs="0" />
              <xsd:element name="au_fname" type="xsd:string" minOccurs="0" />
              <xsd:element name="au_lname" type="xsd:string" minOccurs="0" />
              <xsd:element name="phone" type="xsd:string" minOccurs="0" />
              <xsd:element name="address" type="xsd:string" minOccurs="0" />
              <xsd:element name="city" type="xsd:string" minOccurs="0" />
              <xsd:element name="state" type="xsd:string" minOccurs="0" />
            
```

## Pasos siguientes

Este tutorial muestra los pasos básicos para leer un archivo XML en un conjunto de datos, así como crear un esquema basado en el contenido del archivo XML. Éstas son algunas de las tareas que pueden venir a continuación:

- Editar los datos del conjunto de datos y volver a escribirlos como XML. Para obtener más información, vea [DataSet.WriteXml \(Método\)](#).
- Editar los datos del conjunto de datos y escribirlos en una base de datos. Para obtener más información, vea [Actualizar un origen de datos con un conjunto de datos](#).

## Vea también

[Introducción a conjuntos de datos](#) | [DataSet.ReadXml \(Método\)](#) | [DataSet.WriteXmlSchema \(Método\)](#) | [Depurar código administrado](#) | [Implementar aplicaciones](#) | [XML en Visual Studio](#) | [Tutoriales sobre datos](#)

# Tutoriales sobre los formularios Windows Forms

Los siguientes tutoriales ofrecen instrucciones paso a paso para crear diversas aplicaciones Windows.

## En esta sección

### [Crear una aplicación para Windows con Visual J#](#)

Explica el proceso para escribir y ejecutar un formulario Windows Forms sencillo.

### [Crear un formulario Windows Forms Principal-Detalle con Visual J#](#)

Describe cómo crear una relación entre primario y secundario en un conjunto de datos y mostrar registros relacionados en un formulario Windows Forms.

### [Crear menús contextuales dinámicos en formularios Windows Forms con Visual J#](#)

Explica detalladamente, mediante un ejemplo, cómo crear un único menú contextual para dar servicio a dos o más controles diferentes.

### [Cambiar de estructura de menús en formularios Windows Forms basándose en el estado de una aplicación con Visual J#](#)

Ofrece indicaciones para cambiar entre objetos MainMenu mediante programación.

### [Mostrar la herencia de Visual con Visual J#](#)

Describe cómo se crea un formulario Windows Forms base y cómo se compila en una biblioteca de clases. Se importa la biblioteca de clases a otro proyecto y se crea un nuevo formulario que hereda del formulario base.

### [Crear una interfaz de usuario de varios paneles con formularios Windows Forms con Visual J#](#)

Explica cómo se crea una interfaz de usuario de varios paneles similar a la que se utiliza en Microsoft Outlook.

### [Acceso a datos sencillo en un formulario Windows Forms con Visual J#](#)

Describe los pasos básicos requeridos para tener acceso a datos de SQL Server a través de un conjunto de datos en un formulario Windows Forms de lectura y escritura.

### [Mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

Explica cómo crear un conjunto de datos que contenga registros seleccionados, basándose en criterios que proporcionan los usuarios en tiempo de ejecución en un formulario Windows Forms.

### [Llamar a los servicios Web XML desde los formularios Windows Forms con Visual J#](#)

Explica cómo llamar a métodos de servicios Web desde una aplicación para Windows.

### [Crear una aplicación de servicios de Windows en el Diseñador de componentes con Visual J#](#)

Describe cómo se crea un archivo ejecutable de ejecución larga que carece de interfaz de usuario.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, trabajar con formularios Windows Forms y Web Forms, y realizar tareas relacionadas con datos.

### [Crear aplicaciones para Windows](#)

Sección de Visual Basic/Visual C# que explica cómo crear aplicaciones tradicionales basadas en Windows.

### [Tutoriales de los formularios Windows Forms](#)

Proporciona vínculos a tutoriales adicionales de la documentación de Visual Basic/Visual C#.

# Tutorial: crear una aplicación para Windows con Visual J#

El siguiente tutorial explica el proceso para escribir y ejecutar un formulario Windows Forms sencillo utilizando Visual J#.

## Para crear una aplicación para Windows utilizando Visual J#

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, elija **Proyecto**.

Aparecerá el cuadro de diálogo **Nuevo proyecto**.

2. Seleccione la carpeta **Proyectos de Visual J#** y elija el icono **Aplicación para Windows**.
3. En el cuadro **Nombre**, escriba **JSharpHelloWorld**.
4. Haga clic en **Aceptar**.

La plantilla de proyecto creará una solución y abrirá el Diseñador de Windows Forms.

5. Abra el Cuadro de herramientas y arrastre un control de etiqueta y otro de botón hasta Form1.

Se crearán los controles **label1** y **button1**.

6. Utilice la ventana Propiedades para cambiar la propiedad text de **button1** para que muestre **Haga clic aquí**.
7. Haga doble clic en **button1**.

Se creará el controlador de eventos **button1\_Click** en el archivo de código fuente Form1.jsl. Se abrirá el Editor de código.

8. Reemplace el cuerpo del controlador de eventos con el siguiente código:

```
label1.setText("Hello, world");
```

9. Presione F5 para ejecutar el proyecto.

Se generará y ejecutará el proyecto.

10. Haga clic en el botón **Haga clic aquí** del formulario.

El texto **Label1** cambia a "Hello, World".

## Vea también

[Tutoriales sobre los formularios Windows Forms](#)



# Tutorial: crear un formulario Windows Forms Principal-Detalle con Visual J#

En muchos escenarios de aplicaciones, es posible que desee trabajar con los datos que vienen de una o varias tablas y, con frecuencia, datos de tablas relacionadas. Es decir, trabajar con una relación principal-detalle entre una tabla primaria y una secundaria. Por ejemplo, quizá desee obtener un registro de clientes y ver los pedidos relacionados.

El modelo de base de datos desconectado en Visual Studio le permite trabajar con varias tablas discretas en la aplicación y definir una relación entre ellas. A continuación, puede utilizar la relación para desplazarse a través de registros relacionados.

En este tutorial se muestra un sencillo pero completo ejemplo de trabajo con tablas relacionadas en un conjunto de datos. En el tutorial, creará un formulario Windows Forms que muestra información de editoriales (los registros principales) en un cuadro de lista.

En el tutorial, creará la relación entre las tablas. A continuación, puede utilizar las funciones de enlace de datos del formulario Windows Forms para beneficiarse de esta relación.

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Pubs de SQL Server.

El tutorial está dividido en varias partes más pequeñas:

- Crear el formulario Windows Forms.
- Crear y configurar el conjunto de datos que contiene las dos tablas. Esto incluye la creación de una relación entre las tablas.
- Agregar controles para mostrar datos y enlazarlos a los objetos de datos apropiados.
- Escribir unas pocas líneas de código que llenarán el conjunto de datos cuando se ejecute el formulario.
- Comprobar el formulario.

## Crear el proyecto y el formulario

El primer paso es crear un formulario Windows Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, seleccione **Proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#** y, en el panel **Plantillas**, elija **Aplicación para Windows**.
3. Si ya tiene una solución abierta, seleccione **Cerrar solución**.

**Nota** En proyectos de producción, con mucha frecuencia tendrá varios proyectos en la misma solución. No obstante, en este tutorial, cerrará cualquier solución abierta y creará una nueva junto con el proyecto, por lo que no existirá ninguna interferencia entre lo que hará aquí y los formularios, conjuntos de datos, etc. existentes.

4. Asigne un nombre al proyecto que sea único y cumpla las convenciones de nomenclatura utilizadas. Por ejemplo, podría denominar este proyecto como **Walkthrough\_MasterDetail**.
5. Cuando haya asignado un nombre y especificado una nueva solución, haga clic en **Aceptar**.

Visual Studio crea un proyecto nuevo y muestra un formulario nuevo denominado Form1 en el Diseñador de Windows Forms.

## Crear y configurar un conjunto de datos

Como sucede con la mayoría de los escenarios de acceso a datos en Visual Studio, trabajará con un conjunto de datos. Un conjunto de datos es un contenedor, una caché, que contiene los registros con los que le interesa trabajar.

**Nota** Usar un conjunto de datos es sólo una de las opciones para obtener acceso a datos, y no es la elección ideal en algunos escenarios. El trabajo con datos principal-detalle es un escenario en el que los conjuntos de datos constituyen una buena estrategia de acceso a datos. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos](#).

En este tutorial, agregará un conjunto de datos al formulario. No obstante, no lo hará directamente, sino manualmente, agregándolo al formulario. En su lugar, seguirá los pasos siguientes:

- Crear un adaptador de datos con un asistente. El adaptador contiene instrucciones SQL que se utilizan para leer y escribir información en la base de datos. El asistente le ayuda a definir las instrucciones SQL que necesita. Si es preciso, el asistente también crea una conexión a la base de datos.
- Generar el esquema del conjunto de datos. En este proceso, hará que Visual Studio cree una nueva clase conjunto de datos basándose en las tablas y columnas a las que está obteniendo acceso. Cuando genera la clase conjunto de datos, también agrega una instancia de ella al formulario.

Cuando haya terminado de generar el conjunto de datos, éste contendrá dos tablas, la tabla Publishers y la tabla Titles, las dos de la base de datos Pubs estándar de SQL Server. Las tablas tienen una relación implícita, el campo `pub_id` es la clave principal de la tabla Publishers y una clave externa en la tabla Titles. Convertirá esta relación implícita en una explícita que pueden aprovechar los controles del formulario.

En esta sección del tutorial, realizará determinados pasos que harán que Visual Studio genere el conjunto de datos. Para obtener más información sobre adaptadores, vea [Introducción a los adaptadores de datos](#). Para obtener más información acerca de los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

## Configurar una conexión de datos y un adaptador de datos

Para empezar, cree adaptadores de datos que contengan las instrucciones SQL que se utilizarán para llenar el conjunto de datos más adelante. Como parte de este proceso, defina una conexión para obtener acceso a la base de datos. Configure los adaptadores de datos con el asistente, lo que facilita la creación de las instrucciones SQL necesarias para obtener acceso a los datos.

Necesitará dos adaptadores de datos, uno para obtener los datos de la tabla Publishers y otro para obtener los datos de la tabla Titles.

**Nota** Cuando el asistente haya finalizado, debe continuar en la sección siguiente para generar un conjunto de datos y completar la parte de acceso a datos del formulario.

### Para crear la conexión de datos y los adaptadores de datos

1. Desde la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **OleDbDataAdapter** al formulario.

**Nota** También es posible utilizar el proveedor de datos de SQL (en este caso, el objeto **SqlDataAdapter**), que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa **OleDbDataAdapter** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB. Si en las aplicaciones utiliza SQL Server versión 7.0 o posterior, use el proveedor de datos de SQL.

Se inicia el **Asistente para la configuración del adaptador de datos**, que ayuda a crear la conexión y el adaptador.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Pubs de SQL Server. Para llegar a la segunda página, haga clic en **Siguiente** en la primera página.
  - b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
  - c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT pub_id, pub_name
FROM publishers
```

Escríbala directamente o haga clic en Generador de consultas para obtener ayuda para generar la instrucción.

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Publishers. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

3. Haga clic en **Finalizar** para completar el asistente.

Cuando el asistente haya finalizado, tendrá una conexión (**OleDbConnection1**) que contiene información sobre cómo tener acceso a la base de datos. También tendrá un adaptador de datos (**OleDbDataAdapter1**) que contiene la instrucción SQL para obtener información de la tabla Publishers.

4. Arrastre un segundo objeto **OleDbDataAdapter** al formulario.

El **Asistente para la configuración del adaptador de datos** se inicia de nuevo.

5. Repita el paso 2, con estas diferencias:

- En la segunda página, elija la misma conexión que utilizó o creó la última vez.
- En la cuarta página, cree la siguiente instrucción SQL para tener acceso a la tabla Titles:

```
SELECT title_id, title, pub_id, price
FROM titles
```

**Nota** Debe incluir la columna `pub_id`.

El asistente agrega el objeto **OleDbDataAdapter2** al formulario.

6. Cuando el asistente haya finalizado, genere el conjunto de datos basándose en la consulta SQL que ha creado durante este procedimiento. Para obtener más información, vea la siguiente sección.

## Crear el conjunto de datos

Después de haber establecido los métodos para conectarse a la base de datos y de especificar la información que desea (a través de los comandos SQL de los adaptadores de datos), puede hacer que Visual Studio cree un conjunto de datos. El conjunto de datos es una instancia de la clase **DataSet** basada en un esquema XML correspondiente (archivo .xsd) que describe los elementos de la clase (tablas, columnas y restricciones). Para obtener más información acerca de la relación entre los conjuntos de datos y los esquemas, vea [Introducción al acceso a datos con ADO.NET](#).

### Para generar un conjunto de datos

1. Desde el menú **Datos**, elija **Generar conjunto de datos**.

**Sugerencia** Si no ve el menú **Datos**, haga clic en el formulario; el formulario debe tener el foco para que aparezca el menú.

Se mostrará el cuadro de diálogo **Generar conjunto de datos**.

2. Seleccione la opción **Nuevo** y el nombre del conjunto de datos **dsPublishersTitles**.

En la lista situada bajo **Elegir las tablas que desea agregar al conjunto de datos**, asegúrese de que tanto **publishers** como **titles** están seleccionadas.

3. Marque la casilla **Agregar este conjunto de datos al diseñador** y, a continuación, haga clic en **Aceptar**.

Visual Studio genera una clase de conjunto de datos con tipo (**dsPublishersTitles**) y un esquema que define el conjunto de datos. Verá el nuevo esquema (**dsPublishersTitles.xsd**) en el **Explorador de soluciones**.

**Sugerencia** En el Explorador de soluciones, haga clic en **Mostrar todos los archivos** para ver el archivo .jsl dependiente del archivo de esquema, que contiene el código que define la nueva clase de conjunto de datos.

Por último, Visual Studio agrega una instancia de la nueva clase de conjunto de datos (**dsPublishersTitles1**, observe el número secuencial) al formulario.

## Crear una relación entre tablas del conjunto de datos

El conjunto de datos contiene dos tablas que ya sabe que tienen una relación uno a varios. No obstante, el conjunto de datos es un contenedor pasivo (no es una base de datos real) y no puede utilizar las relaciones implícitas entre las tablas. Por tanto, para convertir la relación en explícita, cree un objeto de relación de datos.

**Nota** Observe que aún no ha creado una consulta única que fusione las tablas Publisher y Titles en la base de datos. En su lugar, cree dos consultas para tener acceso a las dos tablas independientemente. Esto le permite administrar el desplazamiento, las relaciones y las actualizaciones por separado de las dos tablas, lo que le ofrece más control sobre los datos que si hubiera basado las tablas del conjunto de datos en una combinación. Para obtener información más detallada, vea [Introducción a los objetos DataRelation](#).

### Para crear una relación entre tablas

1. En el Explorador de soluciones, haga doble clic en el esquema del conjunto de datos que acaba de crear (denominado **dsPublishersTitles.xsd**).

El **Diseñador XML** se abre en la ventana **Esquema**, lo que muestra las dos tablas del conjunto de datos.

- Desde la ficha **Esquema XML** del Cuadro de herramientas, arrastre un objeto **Relation** a la tabla Titles (la tabla secundaria).

Se abrirá el cuadro de diálogo **Editar relaciones** con los valores predeterminados derivados de las dos tablas.

- Confirme que en el cuadro de diálogo **Editar relación** se han establecido los siguientes valores.

Configuración	Valor
Nombre	<b>publisherstitles</b>  Necesitará saber este nombre más tarde; si lo cambia, asegúrese de que anota el nuevo nombre.
Elemento primario	publishers
Elemento secundario	titles
Campos clave	pub_id
Campo de clave externa	pub_id
Crear sólo una restricción FOREIGN KEY	(Desactivado)

Los valores no especificados anteriormente no son importantes en este tutorial.

- Haga clic en **Aceptar** para cerrar el cuadro de diálogo **Editar relación**.

En el Diseñador XML se muestra un icono de relación entre las dos tablas. Si necesita cambiar la configuración de la relación, puede hacer clic con el botón secundario del *mouse* en la relación y elegir **Editar relación**.

- Guarde el esquema y cierre el Diseñador XML.

Llegados a este punto, tiene configurado todo lo necesario para obtener información de la base de datos y almacenarla en un conjunto de datos. Ya puede agregar controles al formulario para mostrar los datos.

## Agregar controles para mostrar datos

En este tutorial, agregará sólo unos pocos controles: un control **ListBox** para mostrar una lista de editoriales y un control **DataGrid** para mostrar los títulos de una editorial. También puede agregar etiquetas según sea necesario para indicar lo que muestra el formulario.

### Mostrar las editoriales en un control ListBox

El control **ListBox** muestra los nombres de las editoriales. El control es un control de enlace complejo; es decir, puede mostrar varios registros a la vez.

#### Para agregar un control ListBox enlazado al formulario a fin de mostrar editoriales

- Vuelva al formulario predeterminado (Form1) que se abrió cuando creó el proyecto. Si aún no lo ha hecho, cambie al diseñador de formularios haciendo clic en la ficha, en la parte superior de la ventana actual.
- Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **ListBox** al formulario.
- Presione **F4** para mostrar la ventana Propiedades.
- En la propiedad **DataSource**, seleccione **dsPublishersTitles1** como origen de datos.
- En la propiedad **DisplayMember**, muestre la lista desplegable. Seleccione **publishers**, expanda el nodo **publishers** y, a continuación, seleccione **pub\_name**.

Cuando termine, en la propiedad **DisplayMember** debería leerse **publishers.pub\_name**.

Al establecer estas dos propiedades se enlaza el control **ListBox** al campo de nombre de editorial de la tabla Publishers.

- Si lo desea, agregue una etiqueta delante del cuadro de lista para identificarlo.

### Mostrar los títulos en un control DataGrid

Cada registro de título contiene varios bits de información. Por tanto, para mostrar los títulos, quizá desee utilizar un control que no sólo muestre varios registros, sino varias columnas. Una buena opción es el control **DataGrid**.

En este tutorial, conviene configurar la cuadrícula para que sólo muestre los títulos que están relacionados con las editoriales seleccionadas actualmente. Puede utilizar las funciones de enlace de datos de los formularios Windows Forms para hacerlo.

Enlazará la cuadrícula a la tabla Titles (como podría suponer), pero con el objeto de relación de datos que ha creado

anteriormente que establece una relación entre Publishers y Titles. El objeto de relación de datos se expone como una propiedad de la tabla Publishers. A medida que se ejecuta el formulario, siempre que se desplace a una nueva editorial (seleccionándola en el cuadro de lista), el marco de trabajo de enlace de datos del formulario consulta el objeto de relación de datos para devolver los registros de detalle correspondientes.

### Para agregar un control DataGrid enlazado al formulario a fin de mostrar títulos

1. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **DataGrid** al formulario.
2. En la ventana Propiedades, establezca la propiedad **DataSource** en **dsPublishersTitles1**.
3. Establezca la propiedad **DataMember** en **publishers.publishertitles**. Escriba este valor en el campo.

Cuando se establecen estas dos propiedades se enlaza la cuadrícula al objeto de relación, por lo que siempre contiene registros secundarios de la editorial actual.

4. Cambie el tamaño de la cuadrícula para que se puedan ver todas las columnas. Cambie el alto con lo que podrá ver varios registros de títulos.

### Llenar el conjunto de datos

Aunque los controles del formulario están enlazados con el conjunto de datos que creó, dicho conjunto de datos no se rellena automáticamente. En su lugar, debe rellenarlo usted mismo utilizando algunas líneas de código que se ejecutan cuando se inicializa el formulario. Para obtener más información sobre cómo rellenar los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

### Para llenar el conjunto de datos

1. Haga doble clic en el formulario a fin de crear un controlador para el evento **Load** del formulario.
2. En el método, borre el conjunto de datos que ha creado y, a continuación, llame al método **Fill** de los dos adaptadores de datos, pasándole a cada llamada el conjunto de datos que se va a llenar.

En el ejemplo siguiente se muestra la apariencia del método completo.

```
// Visual J#
private void Form1_Load(Object sender, System.EventArgs e)
{
    dsPublishersTitles1.Clear();
    oleDbDataAdapter1.Fill(dsPublishersTitles1);
    oleDbDataAdapter2.Fill(dsPublishersTitles1);
    oleDbConnection1.Close();
}
```

### Pruebas

Ahora puede comprobar el formulario para asegurarse de que muestra los datos de la forma prevista.

### Para comprobar el formulario

1. Presione **F5** para ejecutar el formulario.
2. Cuando se muestre el formulario:
  - Compruebe que el control de cuadro de lista muestra los nombres de las editoriales.
  - Seleccione distintas editoriales y confirme que en la cuadrícula se muestran los títulos correctos (y sólo esos).

### Pasos siguientes

En este tutorial, ha realizado dos tareas: ha creado un conjunto de datos con dos tablas y una relación y ha creado un formulario Windows Forms para mostrar los datos relacionados. Algunos pasos adicionales que podría realizar son los siguientes:

- Personalizar la presentación de columnas en la cuadrícula. Para obtener más información, vea [Dar formato al control DataGrid de formularios Windows Forms en tiempo de diseño](#).
- Utilizar los datos, incluida la relación de datos, en su propio código. (En este tutorial, la arquitectura de enlace de datos del formulario realizó todo el trabajo para resolver los registros secundarios relacionados.) Para obtener más información, vea [Introducción a los objetos DataRelation](#) y [Desplazarse por una relación entre tablas](#).

- Separar el acceso a datos de la interfaz de usuario. En este tutorial, ha creado un formulario que tiene acceso a datos de una forma más o menos directa (a través del conjunto de datos). Un diseño más flexible y fácil de mantener es crear un componente de acceso a datos que controle el acceso a datos. El formulario (es decir, la interfaz de usuario) podría interactuar con el componente según fuera necesario. Varios formularios (y otros componentes) podrían utilizar el mismo componente, lo que elimina la sobrecarga y la redundancia de nuevos diseños de acceso a datos para cada formulario que se cree. Para obtener más información sobre la creación de accesos a datos basados en los componentes, vea [Tutorial: crear una aplicación distribuida con Visual J#](#).

### **Vea también**

[Crear formularios Windows Forms](#) | [Introducción a conjuntos de datos](#) | [Introducción a los objetos DataRelation](#) | [Desplazarse por una relación entre tablas](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: crear menús contextuales dinámicos en formularios Windows Forms con Visual J#

Para ahorrar tiempo y el código implicado en la creación de aplicaciones, varios controles pueden compartir un único objeto de menú contextual. Al disponer de un menú contextual "dinámico" (o menú contextual) que sólo presenta los elementos de menú necesarios para dicho control, se puede reducir el número global de menús contextuales que se necesitan para los controles de la aplicación. En el siguiente tutorial se indica cómo cambiar elementos de menú control a control.

## Crear la aplicación

En los pasos que aparecen a continuación, creará una aplicación para Windows que disponga de un formulario con dos controles. En tiempo de ejecución, al hacer clic con el botón secundario del *mouse* (ratón) en los controles que tengan foco (es decir, que estén seleccionados) se mostrará el menú contextual correspondiente. El control **RadioButton** tendrá dos elementos en su menú contextual y el control **CheckBox** dispondrá de tres.

### Para crear un menú contextual dinámico en un formulario Windows Forms

1. Cree una nueva aplicación Windows. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).
2. Arrastre los controles **CheckBox** y **RadioButton** al formulario desde el Cuadro de herramientas.

Puesto que cabe la posibilidad de que dos (o más) controles compartan un menú contextual, es conveniente que los controles compartan comandos similares, ya que así se reduce la cantidad de veces que es necesario mostrar y ocultar de forma dinámica.

3. Haga doble clic en el componente **ContextMenu** del Cuadro de herramientas para agregarlo al formulario. Éste será el menú contextual compartido.
4. En la ventana Propiedades, establezca la propiedad **ContextMenu** de los controles **CheckBox** y **RadioButton** en **contextMenu1**.
5. En la ventana Propiedades, establezca la propiedad **ThreeState** del control **CheckBox** en **true**.

**Nota** También se puede generar el elemento de menú y el valor de las propiedades **ContextMenu** y **ThreeState** exclusivamente en código. Para obtener información detallada, vea

[Agregar menús contextuales a formularios Windows Forms](#) y

[Establecer las propiedades de controles, documentos y formularios](#). Para obtener información detallada sobre la propiedad **ThreeState**, vea [Introducción al control CheckBox de formularios Windows Forms](#).

6. En el diseñador, haga doble clic en el componente **ContextMenu** para crear un controlador predeterminado para el evento **Popup** de dicho componente. Para obtener información detallada, vea [Crear controladores de eventos en el Diseñador de Windows Forms](#).
7. Inserte código en el controlador de eventos para que realice los siguientes procedimientos:
  - Agregue dos elementos de menú, uno para el estado `Checked` de los controles, y otro para el estado `Unchecked`.
  - Compruebe con una instrucción **if** si el control **CheckBox** es el **SourceControl** del menú contextual. Según el resultado, agregue dinámicamente un tercer elemento de menú para el estado `Indeterminate` del control.

En el siguiente ejemplo, se muestra cómo utilizar el método [Add](#) para establecer la propiedad **Text** del elemento de menú, así como para definir el controlador de eventos asociado a dicho elemento. Para consultar otros métodos **Add** con distintas firmas, vea [Menu.MenuItemCollection.Add \(Método\)](#).

```
// Visual J#
private void ContextMenu1_Popup(Object sender, System.EventArgs e)
{
    // Clear the contents of the context menu.
    contextMenu1.get_MenuItems().Clear();

    // MenuItem 1 - Add menu item for the Checked state.
    contextMenu1.get_MenuItems().Add("Checked", new System.EventHandler(this.Checked_OnClick));
    // MenuItem 2 - Add a menu item for the Unchecked state.
    contextMenu1.get_MenuItems().Add("Unchecked", new System.EventHandler(this.Unchecked_OnClick));
}
```

```
Click));

// Test which control it is.
// MenuItem 3 - If it is the CheckBox, add a menu item for the
// Indeterminate state.
if (contextMenu1.get_SourceControl().Equals(checkBox1))
{
    this.contextMenu1.get_MenuItems().Add("Indeterminate", new System.EventHandler(this
.Indeterminate_OnClick));
}
}
```

8. Cree un controlador de eventos para el primer elemento de menú. Agregue código similar al que se indica a continuación para probar la propiedad **SourceControl** del menú contextual y, en función de ello, establezca la propiedad **Checked** del control **RadioButton** o **CheckBox**:

```
// Visual J#
private void Checked_OnClick(Object sender, System.EventArgs e)
{
    if (contextMenu1.get_SourceControl().Equals( radioButton1))
        radioButton1.set_Checked ( true);
    else if (contextMenu1.get_SourceControl().Equals( checkBox1))
        checkBox1.set_Checked ( true);
}
```

9. Cree un controlador de eventos similar para el segundo elemento de menú. Escriba para el controlador de eventos código similar al que se indica a continuación:

```
// Visual J#
private void Unchecked_OnClick(Object sender, System.EventArgs e)
{
    if (contextMenu1.get_SourceControl().Equals(radioButton1))
        radioButton1.set_Checked ( false);
    else if (contextMenu1.get_SourceControl().Equals( checkBox1))
        checkBox1.set_Checked ( false);
}
```

10. Cree un controlador de eventos similar para el tercer elemento de menú. Escriba un código similar al siguiente para el controlador de eventos, y asegúrese de denominar al evento `Indeterminate_OnClick`:

```
// Visual J#
private void Indeterminate_OnClick(Object sender, System.EventArgs e)
{
    if (contextMenu1.get_SourceControl().Equals( checkBox1))
        checkBox1.set_CheckState ( System.Windows.Forms.CheckState.Indeterminate);
}
```

**Nota** En este ejemplo se utiliza la propiedad **CheckState** para establecer el control **CheckBox** en **Indeterminate** en el controlador de eventos **Indeterminate\_OnClick**.

### Para probar la aplicación

En este momento, ejecutará la aplicación y observará el comportamiento de los elementos del menú contextual al agregarlos y eliminarlos de forma dinámica.

1. Presione **F5** para ejecutar la aplicación. Depúrela si es necesario. Para obtener información detallada, vea [Aspectos básicos sobre depuración](#).
2. Haga clic con el botón secundario del *mouse* (ratón) en **RadioButton** para mostrar el menú contextual.

Tenga en cuenta que existen dos elementos de menú que establecerán el control **RadioButton** en **Checked** o **Unchecked**.



3. Haga clic con el botón secundario del *mouse* (ratón) en **CheckBox** para mostrar el menú contextual.

Tenga en cuenta que existen tres elementos de menú que establecerán el control **CheckBox** en **Checked**, **Unchecked** o **Indeterminate**.

### **Vea también**

[Menús contextuales en formularios Windows Forms](#) | [Copiar elementos de menú desde menús estándar a menús contextuales](#) | [ContextMenu \(Componente, formularios Windows Forms\)](#) | [Implementar aplicaciones y componentes](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: cambiar de estructura de menús en formularios Windows Forms basándose en el estado de una aplicación con Visual J#

La aplicación puede presentar distintos menús para contextos diferentes (o distintos estados de la aplicación). Es posible tener muchos objetos **MainMenu**, cada uno de ellos con distintas opciones de menú para el usuario. Al disponer de varios objetos **MainMenu** para ofrecer al usuario la estructura de menús correcta, se pueden controlar los diversos estados de la aplicación a medida que los usuarios interactúan con la misma.

En este procedimiento, creará una estructura de menús para cuando la aplicación se abra por primera vez y no haya ningún archivo o datos con los que el usuario pueda interactuar. De este modo, la aplicación sólo tendrá un menú Archivo tradicional con los comandos Nuevo, Abrir y Salir. Cuando el usuario selecciona el elemento de menú Nuevo o Abrir, desencadena un cambio en el estado de la aplicación (a modo de demostración, en el ejemplo siguiente aparece un cuadro de mensaje como ayuda visual, pero no se ha creado ni abierto ningún archivo real). En este momento, se muestra una segunda estructura de menús con elementos adicionales (Cerrar y Guardar) dirigidos a una aplicación que tenga un archivo o datos cargados.

## Crear un menú en tiempo de diseño

En los pasos siguientes, diseñará una aplicación para Windows, en el Diseñador de Windows Forms, que cambia las estructuras de menús.

### Para crear un menú en tiempo de diseño

1. Cree un proyecto nuevo de aplicación para Windows.
2. Arrastre un componente **MainMenu** al formulario desde el Cuadro de herramientas.
3. En el Diseñador de menús, cree un elemento de menú de nivel superior con la propiedad **Text** establecida en **&File**, y tres elementos de submenú con las propiedades **Text** establecidas en **&New**, **&Open** y **E&xit**, en ese orden. Para obtener información detallada, vea [Agregar menús y elementos de menú a formularios Windows Forms](#).
4. En el Diseñador de Windows Forms, haga clic en el formulario para asignarle el foco. En la ventana Propiedades, asegúrese de que la propiedad **Menu** esté establecida en el menú que acaba de crear (**mainMenu1**, a menos que se le haya asignado un nombre exclusivo).

### Para enlazar los eventos

1. Cree un controlador de eventos de multidifusión denominado **MenuSelect** para los elementos de menú **Nuevo** y **Abrir**. Para Visual J# debe agregar referencias al controlador de eventos en la función de inicialización del **formulario**:

```
// Visual J#
this.menuItem2.add_Click( new System.EventHandler(this.MenuSelect) );
this.menuItem3.add_Click( new System.EventHandler(this.MenuSelect) );
```

Se trata de un controlador de eventos de multidifusión porque controlará los eventos **Click** de ambos elementos de menú. Para obtener más detalles, vea

[Conectar varios eventos con un único controlador de eventos en formularios Windows Forms](#).

2. Escriba el código siguiente en el controlador de eventos **MenuSelect**:

```
// Visual J#
private void MenuSelect(Object sender, System.EventArgs e)
{
    MessageBox.Show("A file has been opened.", "Instead of a new file, here's a message box.");
    LoadSecondMenu();
}
```

**Nota** `LoadSecondMenu` será un procedimiento para cambiar de menú al hacer clic en alguno de los elementos del menú (**New** u **Open**).

3. En el Editor de código, cree un método similar al siguiente para establecer el menú del formulario en el segundo

componente **MainMenu** creado:

```
// Visual J#
private void LoadSecondMenu()
{
    this.set_Menu ( mainMenu2);
}
```

**Nota** `MainMenu2` hace referencia a un componente **MainMenu** que se agrega al formulario más adelante.

### Para copiar los menús del segundo conjunto

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en el nombre del formulario y elija **Diseñador de vistas** en el menú contextual.
2. En la bandeja de componentes, haga clic en el icono del componente **MainMenu** para que disponga de foco en el formulario.
3. En el Diseñador de menús, haga clic en el elemento de menú de nivel superior que ha creado (**Archivo**). Haga clic con el botón secundario del *mouse* y elija **Copiar**.

**Nota** Con este comando se copian el elemento de menú de nivel superior y todos sus elementos de submenú.

4. Arrastre otro componente **MainMenu** al formulario desde el Cuadro de herramientas.
5. En el Diseñador de menús, haga clic con el botón secundario del *mouse* en el área "Escriba aquí" y elija **Pegar**.

Los elementos de menú seleccionados anteriormente en el primer componente **MainMenu** se pegarán en el segundo.

6. En el Diseñador de menús, haga clic con el botón secundario del *mouse* en el área a la izquierda del elemento de menú **Salir** y elija **Insertar nuevo**. Repita la operación para tener insertados dos elementos de menú nuevos.
7. Establezca la propiedad **Text** de estos dos nuevos elementos de menú en `&Close` y `&Save`, en ese orden, bien en la ventana Propiedades o haciendo clic en el elemento y escribiendo en el espacio que se muestra.

### Para probar la aplicación

- Presione **F5** para ejecutar la aplicación. Depúrela si es necesario. Para obtener información detallada, vea [Aspectos básicos sobre depuración](#).

El formulario tiene un menú que contiene los elementos **Archivo**, **Nuevo**, **Abrir** y **Salir**. Al hacer clic en **Nuevo** o **Abrir** se provoca un evento controlado por el controlador de eventos **MenuSelect**. Este método muestra un cuadro de mensaje y cambia el estado de la aplicación. El cambio de estado de la aplicación viene indicado por la adición de dos elementos de menú, **Close** y **Save**.

En lugar de utilizar el procedimiento anterior, se puede crear toda la aplicación en código. Aunque esto supone renunciar a la facilidad de uso que ofrece el Diseñador de Windows Forms, algunos programadores prefieren este método.

### Crear un menú mediante programación

En los pasos siguientes, diseñará una aplicación para Windows (con idénticos resultados que la anterior) exclusivamente con código que cambia las estructuras de menús.

#### Para crear un menú mediante programación

1. Cree un proyecto nuevo de aplicación para Windows.
2. En un método, cree una instancia del componente **MainMenu** con objetos **MenuItem** correspondientes al estado de la aplicación.

En el siguiente ejemplo se incluye un objeto **MainMenu** para cuando el usuario abre la aplicación por primera vez; sólo tiene un menú **Archivo** tradicional con los comandos **Nuevo**, **Abrir** y **Salir**. Además, el código que aparece a continuación utiliza un método [Add](#) sobrecargado que crea elementos de menú y controladores de eventos asociados a ellos. La conexión de controladores de eventos de forma dinámica es un modo sencillo de indicar a la aplicación que al hacer clic en el comando de menú **Nuevo** o **Abrir** se debe desencadenar un cambio en el estado de la aplicación.

```
// Visual J#
// Create a MainMenu object and a MenuItem object.
```

```

MainMenu mmAppStart;
MenuItem miFile;
public void AppStartMenu()
{
    // Create an instance of the MainMenu object.
    mmAppStart = new MainMenu();

    // Create a top-level menu item and two menu items. Use this
    // overloaded constructor that takes an event handler
    // (MenuSelect) so that later, you can cause the menu selection
    // to change the application state.
    miFile = new MenuItem("&File", new System.EventHandler(MenuSelect));
    miFile.get_MenuItems().Add("&New", new System.EventHandler(MenuSelect));
    miFile.get_MenuItems().Add("&Open", new System.EventHandler(MenuSelect));
    miFile.get_MenuItems().Add("&Exit");

    // Add the top-level menu item to the MainMenu component
    // and set the MainMenu component to be the form's menu.
    mmAppStart.get_MenuItems().Add(miFile);

    // Set the form's menu to the menu you have just created.
    this.set_Menu ( mmAppStart);
}

```

### Para crear el segundo conjunto de menús

1. En un segundo método, cree una segunda instancia de un componente **MainMenu** con **MenuItems** correspondientes a un segundo estado de la aplicación.

Por ejemplo, una vez que el usuario ha abierto un archivo, se pueden exponer comandos para cerrarlo y guardarlo. Es posible copiar la estructura de menús original para utilizarla como base para esta estructura de menús y conservar los elementos de menú **Archivo**, **Nuevo**, **Abrir** y **Salir**. Para obtener información detallada, vea [Copiar elementos en menús de formularios Windows Forms](#).

```

// Visual J#
// Create the second MainMenu object
private MainMenu mmFileLoadedMenu;
private void FileLoadedMenu()
{
    // Clone first menu
    mmFileLoadedMenu = mmAppStart.CloneMenu();

    // Add Close and Save menu items to the new menu's File menu items,
    // using the Add method to specify their order within the collection.
    MenuItem miFile = mmFileLoadedMenu.get_MenuItems().get_Item(0);
    miFile.get_MenuItems().Add(2, new MenuItem("&Close"));
    miFile.get_MenuItems().Add(3, new MenuItem("&Save"));

    // Assign the newly-created MainMenu object to the form.
    this.set_Menu(mmFileLoadedMenu);
}

```

2. Agregue una línea de código al constructor `Form1`, después de la llamada al método `InitializeComponent`, para llamar al método `AppStartMenu` que se ha creado anteriormente:

```

// Visual J#
AppStartMenu();

```

3. Cree un controlador de eventos dentro de la clase, para cambiar la propiedad `Menu` del formulario a `FileLoadedMenu`.

En lugar de cargar un archivo, el código siguiente abre un cuadro de mensaje para indicar un cambio en el estado de la aplicación. Para obtener información detallada, vea [MessageBox \(Clase\)](#). El código llama al método `AppStartMenu` escrito anteriormente, que a su vez crea el segundo objeto **MainMenu** y lo establece como menú del formulario.

```
// Visual J#
protected void MenuSelect(Object sender, System.EventArgs e)
{
    MessageBox.Show("A file has been opened.", "Instead of a new file, here's a message box.");
    FileLoadedMenu();
}
```

### Para probar la aplicación

- Presione **F5** para ejecutar la aplicación. Depúrela si es necesario. Para obtener información detallada, vea [Aspectos básicos sobre depuración](#).

El formulario tiene un menú que contiene los elementos **Archivo**, **Nuevo**, **Abrir** y **Salir**. Al hacer clic en **Nuevo** o **Abrir** se provoca un evento controlado por el controlador de eventos **MenuSelect**. Este método muestra un **MessageBox** y cambia el estado de la aplicación. El cambio de estado de la aplicación viene indicado por la adición de dos elementos de menú, **Close** y **Save**.

### Vea también

[Introducción al componente MainMenu de formularios Windows Forms](#) | [Menús en formularios Windows Forms](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: mostrar la herencia de Visual con Visual J#

La herencia visual permite ver los controles del formulario base y agregar controles nuevos. En este tutorial aprenderá a crear un formulario base y a compilarlo en una biblioteca de clases. Se importa la biblioteca de clases a otro proyecto y se crea un nuevo formulario que hereda del formulario base. Durante este tutorial aprenderá a:

- Crear un proyecto de biblioteca de clases que contenga un formulario base.
- Agregar un botón con propiedades que las clases derivadas del formulario base puedan modificar.
- Agregar un botón que no puedan modificar los herederos del formulario base.
- Crear un proyecto que contenga un formulario que herede de **BaseForm**.

Por último, en el tutorial se mostrará la diferencia que existe entre controles privados y protegidos de un formulario heredado.

## Pasos del escenario

El primer paso es crear el formulario base.

### Para crear un proyecto de biblioteca de clases que contenga un formulario base

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, seleccione **Proyecto** para abrir el cuadro de diálogo **Nuevo proyecto**.
2. Cree una aplicación para Windows denominada **BaseFormLibrary**. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).
3. Para crear una biblioteca de clases en lugar de una aplicación convencional para Windows, haga clic con el botón secundario del *mouse* (ratón) en el nodo del proyecto **BaseFormLibrary** en el Explorador de soluciones y seleccione **Propiedades**.
4. En las propiedades del proyecto, cambie el tipo de resultados de **Aplicación para Windows** a **Biblioteca de clases** y haga clic en **Aceptar**.
5. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto y los archivos en la ubicación predeterminada.

Los dos procedimientos siguientes sirven para agregar botones al formulario base. Para ilustrar la herencia visual, se otorgará a los botones distintos niveles de acceso estableciendo sus propiedades **Modifiers**.

### Para agregar un botón que los herederos del formulario base pueden modificar

1. En la ficha **Windows Forms** del Cuadro de herramientas, haga doble clic en **Button** para agregar un botón al formulario. Utilice el *mouse* para colocar el botón y cambiar su tamaño.
2. En la ventana Propiedades, establezca las propiedades siguientes del botón:
  - Establezca la propiedad **Text** en **Say Hello**.
  - Establezca el valor **btnProtected** en la propiedad **Name**.
  - Establezca la propiedad **Modifiers** en **Protected**. Esto permite a los formularios que heredan de **Form1** modificar las propiedades de **btnProtected**.
3. Haga doble clic en el botón **Say Hello** para agregar un controlador de eventos para el evento **Click**.
4. Agregue la siguiente línea de código al controlador de eventos:

```
// Visual J#
MessageBox.Show("Hello, World!");
```

### Para agregar un botón que no pueden modificar los herederos del formulario base

1. Cambie a la vista Diseño presionando MAYÚS+F7.
2. Agregue un segundo botón y establezca sus propiedades como se indica a continuación:
  - Establezca la propiedad **Text** en **Say Goodbye**.
  - Establezca el valor **btnPrivate** en la propiedad **Name**.
  - Establezca la propiedad **Modifiers** en **Private**. Esto impide a los formularios que heredan de **Form1** modificar las propiedades de **btnPrivate**.
3. Haga doble clic en el botón **Say Goodbye** para agregar un controlador de eventos para el evento **Click**. Introduzca la siguiente línea de código en el procedimiento de evento:

```
// Visual J#  
MessageBox.Show ("Goodbye!");
```

4. En el menú **Generar**, elija **Generar BaseFormLibrary** para generar la biblioteca de clases.

Una vez generada la biblioteca, se puede crear un nuevo proyecto que herede del formulario que se acaba de crear.

#### Para crear un proyecto que contenga un formulario que herede del formulario base

1. En el menú **Archivo**, elija **Agregar proyecto** y, a continuación, seleccione **Nuevo proyecto** para abrir el cuadro de diálogo **Nuevo proyecto**.
2. Cree una aplicación para Windows denominada **InheritanceTest**. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).

#### Para agregar un formulario heredado

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el proyecto **InheritanceTest**, seleccione **Agregar** y, a continuación, **Formulario heredado**.
2. En el cuadro de diálogo **Agregar nuevo elemento**, compruebe que la opción **Formulario heredado** está seleccionada y haga clic en **Abrir**.
3. En el cuadro de diálogo **Selector de herencia**, seleccione **Form1** en el proyecto **BaseFormLibrary** como formulario del que se va a heredar y haga clic en **Aceptar**.

Con esto se crea un formulario en el proyecto **InheritanceTest** que deriva del formulario de **BaseFormLibrary**.

4. Haga doble clic para abrir el formulario heredado en el Diseñador de Windows Forms, si aún no está abierto.

En el Diseñador de Windows Forms, los botones heredados tienen un glifo (🔗) en la esquina superior para indicar que son heredados.

5. Seleccione el botón **Say Hello** y observe los controladores de tamaño. Los herederos no pueden mover este botón, cambiar su título o tamaño ni realizar ninguna otra modificación, puesto que está protegido.
6. Seleccione el botón **Say Goodbye** privado y observe que no tiene controladores de tamaño. Además, en la ventana Propiedades, las propiedades de dicho botón aparecen atenuadas para indicar que no se pueden modificar. Por último, coloque el puntero del *mouse* sobre el botón; aparecerá información sobre herramientas que indica cómo se ha heredado el control.
7. Haga clic con el botón secundario del *mouse* en el proyecto **InheritanceTest** en el Explorador de soluciones y seleccione **Establecer como proyecto de inicio**.
8. Presione **F7** para cambiar a la vista Código. Agregue el método siguiente al formulario heredado (**Form2**). Éste es el punto de entrada de la aplicación **InheritanceTest**.

```
// The main entry point for the application.  
/** @attribute System.STAThreadAttribute() */  
public static void main(String[] args)  
{  
    Application.Run(new Form2());  
}
```

9. Haga clic con el botón secundario del *mouse* en el proyecto **InheritanceTest** en el Explorador de soluciones y, a continuación, seleccione **Propiedades**. En el cuadro de diálogo **Páginas de propiedades de InheritanceTest**, establezca que el **Objeto inicial** sea el formulario heredado (probablemente **Form2**).
10. Presione **F5** para ejecutar la aplicación y observe el comportamiento del formulario heredado.

#### Exploración adicional

La herencia de los controles de usuario funciona de forma similar. Abra un nuevo proyecto de biblioteca de clases y agregue un control de usuario. Coloque controles constituyentes en el proyecto y compílelo. Abra otro proyecto de biblioteca de clases nuevo y agregue una referencia a la biblioteca de clases compilada. Pruebe también a agregar al proyecto un control heredado (mediante el cuadro de diálogo Agregar nuevo elemento) y utilice el Selector de herencia. Agregue un control de usuario y cambie la instrucción de herencia (**extends** en Visual J#). Para obtener más información, vea [Heredar formularios Windows Forms](#).

**Vea también**

[Heredar formularios Windows Forms](#) | [Herencia en formularios Windows Forms](#) | [Formularios Windows Forms](#) | [Tutoriales sobre los formularios Windows Forms](#)



# Tutorial: crear una interfaz de usuario de varios paneles con formularios Windows Forms con Visual J#

El control **Splitter** de formularios Windows Forms proporciona un medio sencillo para cambiar el tamaño de una interfaz de usuario compleja. Al establecer la propiedad **Dock** y el orden z (ubicación de los controles en el eje z del formulario) de los controles, se puede crear una interfaz de usuario intuitiva y de fácil utilización.

**Nota** También se puede manipular el orden z de los controles haciendo clic con el botón secundario del *mouse* (ratón) y eligiendo **BringToFront** o **SendToBack**. Para obtener más información acerca del orden z de los controles en los formularios Windows Forms, vea [Disponer objetos en capas en formularios Windows Forms](#).

Además, cuando se acoplan controles en un control desplazable, resulta conveniente agregar un control desplazable secundario como **Panel** para que contenga cualquier otro control que pueda requerir desplazamiento. El control **Panel** secundario debería agregarse al control desplazable, establecerse su propiedad **Dock** en **DockStyle.Fill**, y su propiedad **AutoScroll** establecerse en **true**. La propiedad **AutoScroll** del control desplazable primario debería establecerse en **false**.

Este ejemplo crea una interfaz de usuario de varios paneles similar a la que se utiliza en Microsoft Outlook. El ejemplo se concentra en las técnicas para disponer el control **Splitter** y los demás controles en el formulario, no en agregar funcionalidad para que la aplicación sea útil para un fin determinado. Muestra un control **TreeView** acoplado al lado izquierdo del formulario. El lado derecho del formulario contiene un control **Panel** con un control **ListView** encima de un control **RichTextBox**. La interfaz de usuario la administran dos controles divisores, que permiten cambiar el tamaño de los demás controles de forma independiente. Los controles **TreeView** y **ListView** se utilizan para mostrar datos, mientras que el control **Panel** mantiene el comportamiento correcto en el cambio de tamaño. Puede adaptar las técnicas de este procedimiento para crear interfaces de usuario personalizadas propias.

## Crear la interfaz de usuario en tiempo de diseño

### Para crear una interfaz de usuario de estilo Outlook en tiempo de diseño

1. Cree un proyecto nuevo de aplicación para Windows. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).
2. Arrastre un control **TreeView** desde el **Cuadro de herramientas** hasta el formulario. En la ventana **Propiedades**, establezca la propiedad **Dock** en **Left** haciendo clic en el panel izquierdo del editor de valor que aparece cuando se hace clic en la flecha hacia abajo.
3. Arrastre un control **Splitter** desde el **Cuadro de herramientas** hasta el formulario. Se acoplará automáticamente al borde derecho del control **TreeView**.
4. Arrastre un control **Panel** desde el **Cuadro de herramientas** hasta el formulario. En la ventana **Propiedades**, establezca la propiedad **Dock** en **Fill** haciendo clic en el panel central del editor de valor que aparece cuando se hace clic en la flecha hacia abajo. El panel llenará por completo el lado derecho del formulario.
5. Arrastre un control **ListView** desde el **Cuadro de herramientas** hasta el control **Panel** que agregó al formulario. Establezca la propiedad **Dock** del control **ListView** en **Top**.
6. Arrastre un control **Splitter** desde el **Cuadro de herramientas** hasta el control **Panel** que agregó al formulario. En la ventana **Propiedades**, establezca la propiedad **Dock** en **Top** haciendo clic en el panel superior del editor de valor que aparece cuando se hace clic en la flecha hacia abajo. Esto lo acoplará en la parte inferior del control **ListView**.
7. Arrastre un control **RichTextBox** desde el **Cuadro de herramientas** hasta el control **Panel**. Establezca la propiedad **Dock** del control **RichTextBox** en **Fill**.
8. Presione **F5** para ejecutar la aplicación.

El formulario muestra una interfaz de usuario de tres partes, similar a la de Microsoft Outlook. Observe que, cuando se arrastra el *mouse* sobre cualquiera de los controles **Splitter**, se puede cambiar el tamaño de las ventanas.

## Crear la interfaz de usuario mediante programación

### Para crear una interfaz de usuario de estilo Outlook mediante programación

1. Cree un proyecto nuevo de aplicación para Windows. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).
2. Declare cada uno de los controles comprendidos en la interfaz de usuario. En este ejemplo, utilice los controles **TreeView**, **ListView**, **Panel**, **Splitter** y **RichTextBox** para obtener la apariencia de la interfaz de usuario de Microsoft Outlook.

```
// Visual J#
private System.Windows.Forms.TreeView treeView1;
private System.Windows.Forms.ListView listView1;
private System.Windows.Forms.RichTextBox richTextBox1;
private System.Windows.Forms.Splitter splitter2;
private System.Windows.Forms.Splitter splitter1;
private System.Windows.Forms.Panel panel1;
```

3. Cree un procedimiento que defina la interfaz de usuario. El código siguiente establece las propiedades para que el formulario se asemeje a la interfaz de usuario de Microsoft Outlook. Sin embargo, si utiliza otros controles o los acopla de forma diferente, es igual de fácil crear otras interfaces de usuario que tengan la misma flexibilidad.

```
// Visual J#
public void OutlookUI()
{
    // Create an instance of each control being used.
    treeView1 = new System.Windows.Forms.TreeView();
    listView1 = new System.Windows.Forms.ListView();
    richTextBox1 = new System.Windows.Forms.RichTextBox();
    splitter2 = new System.Windows.Forms.Splitter();
    splitter1 = new System.Windows.Forms.Splitter();
    panel1 = new System.Windows.Forms.Panel();
    // Insert code here to hook up event methods.

    // Set properties of TreeView control.
    treeView1.set_Dock ( System.Windows.Forms.DockStyle.Left);
    treeView1.set_Width ( this.get_ClientSize().get_Width()/3);
    treeView1.set_TabIndex ( 0);
    treeView1.get_Nodes().Add("treeView");

    // Set properties of ListView control.
    listView1.set_Dock ( System.Windows.Forms.DockStyle.Top);
    listView1.set_Height ( this.get_ClientSize().get_Height() * 2 / 3);
    listView1.set_TabIndex ( 0);
    listView1.get_Items().Add("listView");

    // Set properties of RichTextBox control.
    richTextBox1.set_Dock ( System.Windows.Forms.DockStyle.Fill);
    richTextBox1.set_TabIndex ( 2);
    richTextBox1.set_Text ( "richTextBox1");

    // Set properties of Panel's Splitter control.
    splitter2.set_Dock ( System.Windows.Forms.DockStyle.Top);
    // Width is irrelevant if splitter is docked to Top.
    splitter2.set_Height ( 3);
    // Use a different color to distinguish the two splitters.
    splitter2.set_BackColor (Color.get_Blue());
    splitter2.set_TabIndex ( 1);
    // Set TabStop to false for ease of use when negotiating
    // the user interface.
    splitter2.set_TabStop ( false);

    // Set properties of Form's Splitter control.
    splitter1.set_Location (new System.Drawing.Point(121, 0));
    splitter1.set_Size ( new System.Drawing.Size(3, 273));
    splitter1.set_BackColor ( Color.get_Red());
    splitter1.set_TabIndex ( 1);
    // Set TabStop to false for ease of use when negotiating
    // the user interface.
```

```
splitter1. set_TabStop (false);

// Add the appropriate controls to the Panel.
// The order of the controls in this call is critical.
panel1.get_Controls().AddRange(new System.Windows.Forms.Control[]
    {richTextBox1, splitter2, listView1});

// Set properties of Panel control.
panel1. set_Dock ( System.Windows.Forms.DockStyle.Fill);
panel1. set_TabIndex (2);

// Add the rest of the controls to the form.
// The order of the controls in this call is critical.
this.get_Controls().AddRange(new System.Windows.Forms.Control[]
    {panel1, splitter1, treeView1});
this. set_Text ( "Intricate UI Example");
}
```

4. Agregue esta línea de código al constructor de la clase de formulario.

```
// Visual J#
// Add this to the form class's constructor.
OutlookUI();
```

5. Presione **F5** para ejecutar la aplicación.

### **Vea también**

[Introducción al control Splitter de formularios Windows Forms](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: acceso a datos sencillo en un formulario Windows Forms con Visual J#

Uno de los escenarios más habituales en el desarrollo de aplicaciones es mostrar datos en un formulario. Este tutorial presenta un formulario Windows Forms sencillo que muestra datos de una sola tabla en una cuadrícula de datos. La cuadrícula es editable y podrá realizar cambios en los datos y actualizar la base de datos. Aunque el resultado no es complejo, el tutorial muestra muchos de los procedimientos básicos que utilizará cuando tenga acceso a datos con formularios.

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Pubs de SQL Server.

El tutorial está dividido en varias partes más pequeñas:

- Crear el formulario Windows Forms.
- Crear y configurar el conjunto de datos con respecto al cual se enlazará el formulario. Esto incluye crear una consulta que llene el conjunto de datos a partir de la base de datos.
- Agregar el control **DataGrid** al formulario y enlazarlo a los datos.
- Agregar código para llenar el conjunto de datos.
- Agregar código que envíe cambios del conjunto de datos de vuelta a la base de datos.

Cuando haya finalizado el tutorial, tendrá un formulario que se parecerá al siguiente.

au_id	au_lname	au_fname	city	state	phone	contract
172-32-1176	White	Johnson	Menlo Park	CA	408 496-7223	<input checked="" type="checkbox"/>
213-46-8915	Green	Marjorie	Oakland	CA	415 986-7020	<input checked="" type="checkbox"/>
238-95-7766	Carson	Cheryl	Berkeley	CA	415 548-7723	<input checked="" type="checkbox"/>
267-41-2394	O'Leary	Michael	San Jose	CA	408 286-2428	<input checked="" type="checkbox"/>
274-80-9391	Straight	Dean	Oakland	CA	415 834-2919	<input checked="" type="checkbox"/>
341-22-1782	Smith	Meander	Lawrence	KS	913 843-0462	<input type="checkbox"/>
409-56-7008	Bennet	Abraham	Berkeley	CA	415 658-9932	<input checked="" type="checkbox"/>
427-17-2319	Dull	Ann	Palo Alto	CA	415 836-7128	<input checked="" type="checkbox"/>
472-27-2349	Gringlesby	Burt	Covelo	CA	707 938-6445	<input checked="" type="checkbox"/>
486-29-1786	Locksley	Charlene	San Francisco	CA	415 585-4620	<input checked="" type="checkbox"/>
527-72-3246	Greene	Morningstar	Nashville	TN	615 297-2723	<input type="checkbox"/>

## Crear el proyecto y el formulario

El primer paso es crear un formulario Windows Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, seleccione **Proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#** y, en el panel **Plantillas**, elija **Aplicación para Windows**.
3. Si ya tiene una solución abierta, seleccione **Cerrar solución**.

**Nota** En proyectos de producción, con mucha frecuencia tendrá varios proyectos en la misma solución. No obstante, en este tutorial, cerrará cualquier solución abierta y creará una nueva junto con el proyecto, por lo que no existirá ninguna interferencia entre lo que hará aquí y los formularios, conjuntos de datos, etc. existentes.

4. Asigne un nombre al proyecto que sea único y cumpla las convenciones de nomenclatura utilizadas. Por ejemplo, podría denominar este proyecto como **Walkthrough\_Simple1**.
5. Cuando haya asignado un nombre y especificado una nueva solución, haga clic en **Aceptar**.

Visual Studio crea un proyecto nuevo y muestra un formulario nuevo en el Diseñador de Windows Forms.

## Crear y configurar un conjunto de datos

Como sucede con la mayoría de los escenarios de acceso a datos en las aplicaciones Windows Forms, trabajará con un conjunto de datos. Un conjunto de datos es un contenedor, una caché, que contiene los registros con los que le interesa trabajar.

**Nota** Usar un conjunto de datos es sólo una de las opciones para obtener acceso a datos, y no es la elección ideal en algunos escenarios. No obstante, los conjuntos de datos son normalmente la elección correcta en las aplicaciones Windows Forms, por lo que usará uno de ellos en este tutorial. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos](#).

En este tutorial, agregará un conjunto de datos al formulario. Sin embargo, ya que el conjunto de datos aún no existe, no lo agregará manualmente al formulario. En su lugar, seguirá los pasos siguientes:

- Crear un adaptador de datos con un asistente. El adaptador contiene instrucciones SQL que se utilizan para leer y escribir información en la base de datos. El asistente le ayuda a definir las instrucciones SQL que necesita. Si es preciso, el asistente también crea una conexión a la base de datos.
- Generar el esquema del conjunto de datos. En este proceso, hará que Visual Studio cree una nueva clase de conjunto de datos con tipo basándose en las tablas y columnas a las que está obteniendo acceso. Cuando genera la clase conjunto de datos, también agrega una instancia de ella al formulario.

Es importante que siga todos los procedimientos de esta sección. En caso contrario, el formulario no tendrá el conjunto de datos que se va a utilizar en las subsiguientes partes del tutorial.

Para obtener más información sobre adaptadores, vea [Introducción a los adaptadores de datos](#). Para obtener más información acerca de los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

## Configurar una conexión de datos y un adaptador de datos

Para empezar, cree un adaptador de datos que contenga la instrucción SQL que se utilizará para llenar el conjunto de datos más adelante. Como parte de este proceso, defina una conexión para obtener acceso a la base de datos. Configure el adaptador de datos con el asistente, lo que facilita la creación de las instrucciones SQL necesarias para obtener acceso a los datos.

**Nota** Cuando el asistente haya finalizado, debe continuar en la sección siguiente para generar un conjunto de datos y completar la parte de acceso a datos del formulario.

### Para crear la conexión de datos y el adaptador de datos

1. Desde la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **OleDbDataAdapter** al formulario.

**Nota** También es posible usar **SqlDataAdapter**, que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa **OleDbDataAdapter** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB.

Se inicia el **Asistente para la configuración del adaptador de datos**, que ayuda a crear la conexión y el adaptador.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Pubs de SQL Server. Para llegar a la segunda página, haga clic en **Siguiente** en la primera página.
  - b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
  - c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT au_id, au_lname, au_fname, city, state, phone, contract
FROM authors
```

Para obtener ayuda para generar la instrucción SQL, haga clic en **Generador de consultas** para iniciar el **Generador de consultas**.

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Authors. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

El asistente crea una conexión (**OleDbConnection1**) que contiene información sobre cómo tener acceso a la base de datos. También tendrá un adaptador de datos (**OleDbDataAdapter1**) que contiene una consulta que define la tabla y las columnas de la base de datos a la que desea tener acceso.

3. Cuando el asistente haya finalizado, genere el conjunto de datos basándose en la consulta SQL que ha creado durante este procedimiento. Para obtener más información, vea la siguiente sección.

## Crear el conjunto de datos

Después de haber establecido los métodos para conectarse a la base de datos y de especificar la información que desea (a través del comando SQL del adaptador de datos), puede hacer que Visual Studio cree un conjunto de datos. Visual Studio puede generar el conjunto de datos automáticamente basándose en la consulta que ha especificado para el adaptador de datos. El conjunto de datos es una instancia de la clase **DataSet** basada en un esquema XML correspondiente (archivo .xsd) que describe los elementos de la clase (tabla, columnas y restricciones). Para obtener más información acerca de la relación entre los conjuntos de datos y los esquemas, vea [Introducción al acceso a datos con ADO.NET](#).

### Para generar un conjunto de datos

1. Desde el menú **Datos**, elija **Generar conjunto de datos**.

**Sugerencia** Si no ve el menú **Datos**, haga clic en el formulario; el formulario debe tener el foco para que aparezca el menú.

Se mostrará el cuadro de diálogo **Generar conjunto de datos**.

2. Seleccione la opción **Nuevo** y el nombre del conjunto de datos **dsAuthors**.

En la lista situada bajo **Elegir las tablas que desea agregar al conjunto de datos**, la tabla Authors debería aparecer seleccionada.

3. Asegúrese de que la casilla **Agregar este conjunto de datos al diseñador** está seleccionada y haga clic en **Aceptar**.

Visual Studio genera una clase de conjunto de datos con tipo (**dsAuthors**) y un esquema que define el conjunto de datos. Verá el nuevo esquema (**dsAuthors.xsd**) en el **Explorador de soluciones**.

**Sugerencia** En el Explorador de soluciones, haga clic en **Mostrar todos los archivos** para ver el archivo .jsl dependiente del archivo de esquema, que contiene el código que define la nueva clase de conjunto de datos.

Por último, Visual Studio agrega una instancia de la nueva clase de conjunto de datos (**dsAuthors1**) al formulario.

Llegados a este punto, tiene configurado todo lo necesario para obtener información de la base de datos y almacenarla en un conjunto de datos. Ya puede crear un formulario que muestre los datos.

### Agregar un control DataGridView para mostrar los datos

En este tutorial, agregará un único control, **DataGridView**, que puede mostrar todos los registros del conjunto de datos simultáneamente. Una alternativa sería utilizar controles individuales como cuadros de texto para mostrar un registro cada vez. Esa estrategia requiere que agregue opciones de desplazamiento al formulario. En consecuencia, por sencillez, utilizará una cuadrícula de datos.

**Nota** Para obtener un ejemplo de cómo utilizar cuadros de texto individuales para mostrar registros de un conjunto de datos, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#).

La cuadrícula de datos debe enlazarse al conjunto de datos para mostrar los datos.

#### Para agregar un control DataGridView al formulario

1. Si aún no lo ha hecho, cambie al diseñador de formularios haciendo clic en la ficha, en la parte superior de la ventana actual.
2. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **DataGridView** al formulario.
3. Presione **F4** para mostrar la ventana Propiedades.
4. En la propiedad **DataSource**, seleccione **DsAuthors1** como origen de datos. No elija **DsAuthors1.authors**.
5. En la propiedad **DataMember**, seleccione **authors**.

La configuración de estas dos propiedades enlaza la tabla de datos Authors del conjunto de datos **DsAuthors1** a la cuadrícula.

6. Cambie el tamaño de la cuadrícula para que se puedan ver todas las columnas. Cambie el alto de forma que pueda ver varios registros de autores.

### Llenar el control DataGridView

Aunque la cuadrícula de datos está enlazada al conjunto de datos creado, el propio conjunto de datos no se llena de datos automáticamente. En su lugar, debe llenar el conjunto de datos usted mismo llamando a un método de adaptador de datos. Para obtener más información sobre cómo rellenar los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

#### Para llenar el control DataGridView

1. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **Button** al formulario.
2. Llame al botón **btnLoad** y cambie el título estableciendo la propiedad **Text** en **Cargar**.
3. Haga doble clic en el botón para crear un método de control de eventos para su evento **Click**.

4. En el método, borre el conjunto de datos que ha creado y, a continuación, llame al método **Fill** del adaptador de datos, pasándole el conjunto de datos que desea llenar.

En el ejemplo siguiente se muestra la apariencia del método completo.

```
// Visual J#
private void btnLoad_Click(Object sender, System.EventArgs e)
{
    dsAuthors1.Clear();
    oleDbDataAdapter1.Fill(dsAuthors1);
    oleDbConnection1.Close();
}
```

## Actualizar la base de datos

Cuando los usuarios hacen un cambio en la cuadrícula, el control guarda automáticamente el registro actualizado en el conjunto de datos. En formularios Windows Forms, la arquitectura de enlace de datos escribe los valores de los controles enlazados a datos en las filas de datos a los que están enlazados.

**Nota** En páginas de formularios Web Forms, el enlace de datos funciona de diferente manera. Un ejemplo paso a paso del enlace de datos en las páginas de formularios Web Forms puede consultarse en el

[Tutorial: Mostrar datos en una página de formularios con Visual J# y](#)

[Tutorial: Actualizar datos mediante una consulta de actualización de bases de datos en los formularios Web Forms con Visual J#.](#)

No obstante, cuando se trabaja con un conjunto de datos, las actualizaciones requieren dos fases. Después de que los datos están en el conjunto de datos, aún tiene que enviarlos del conjunto de datos a la base de datos. El adaptador de datos puede hacerlo con su método **Update**, que examina cada registro de la tabla de datos especificada del conjunto de datos y, si se ha cambiado algún registro, envía el comando Update, Insert o Delete apropiado a la base de datos. Para obtener más información, vea [Introducción a las actualizaciones de conjuntos de datos](#).

En este tutorial, agregará un botón al formulario, que los usuarios pueden presionar cuando deseen enviar actualizaciones a la base de datos.

### Para actualizar la base de datos

1. Desde la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **Button** al formulario.
2. Llame al botón **btnUpdate** y cambie el título estableciendo la propiedad **Text** en **Guardar cambios en la base de datos**.
3. Haga doble clic en el botón para crear un método de control de eventos para su evento **Click**.
4. En el método, llame al método **Update** del adaptador de datos, pasándole el conjunto de datos que contiene las actualizaciones que desea enviar a la base de datos. Utilice el objeto **MessageBox** para mostrar un texto de confirmación.

En el ejemplo siguiente se muestra la apariencia del método completo.

```
//Visual J#
private void btnUpdate_Click(Object sender, System.EventArgs e)
{
    oleDbDataAdapter1.Update(dsAuthors1);
    oleDbConnection1.Close();
    MessageBox.Show("Database updated!");
}
```

## Pruebas

Ahora puede comprobar el formulario para asegurarse de que muestra los datos de autores en la cuadrícula y que los usuarios pueden realizar actualizaciones.

### Para comprobar el formulario

1. Presione **F5** para ejecutar el formulario.
2. Cuando se muestre el formulario, haga clic en el botón **Cargar**.

En la cuadrícula se muestra una lista de autores.

3. Haga un cambio en un registro de la cuadrícula.

Cuando se desplaza a otro registro de la cuadrícula, se conserva el cambio. Recuerde el cambio.

4. Haga clic en el botón **Cargar**.

Esto vuelve a cargar el conjunto de datos desde la base de datos y actualiza la cuadrícula. Tenga en cuenta que el cambio que ha realizado en el Paso 3 no se ha conservado, porque no guardó los cambios del conjunto de datos en la base de datos.

5. Haga de nuevo un cambio en un registro de la cuadrícula.

6. Haga clic en el botón **Guardar cambios en la base de datos**.

Verá que se muestra un cuadro de mensaje, pero que no hay ningún cambio en la cuadrícula.

7. Haga clic en el botón **Cargar** otra vez para volver a cargar los datos de la base de datos.

Ahora, se conserva el cambio que ha realizado en el Paso 5, porque los datos se han guardado en la base de datos.

## Pasos siguientes

En este tutorial se han mostrado los pasos básicos relacionados con la presentación de datos en un formulario. Algunas mejoras que podría realizar en el formulario en este tutorial incluyen:

- Dar formato a la cuadrícula cambiando su color, fuente, etc.
- Hacer que la cuadrícula muestre datos sin que el usuario la cargue explícitamente. Para ello, agregue un método **Fill** al constructor de formularios y quite el botón **Cargar**.
- Mostrar sólo la información seleccionada en la cuadrícula. En muchos casos, basará la presentación en información que el usuario proporciona en tiempo de ejecución (por ejemplo, podría mostrar sólo los autores de una ciudad determinada). Para ello, cree una consulta parametrizada. Para obtener más información, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)
- Separar el acceso a datos de la interfaz de usuario. En este tutorial, ha creado un formulario que tiene acceso a datos de una forma más o menos directa (a través del conjunto de datos). Un diseño más flexible y fácil de mantener es crear un componente de acceso a datos que controle el acceso a datos. El formulario (es decir, la interfaz de usuario) podría interactuar con el componente según fuera necesario. Varios formularios (y otros componentes) podrían utilizar el mismo componente, lo que elimina la sobrecarga y la redundancia de nuevos diseños de acceso a datos para cada formulario que se cree. Para obtener más información sobre la creación de accesos a datos basados en los componentes, vea [Tutorial: crear una aplicación distribuida con Visual J#](#).

## Vea también

[Tutoriales sobre los formularios Windows Forms](#)



# Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#

Un escenario habitual consiste en mostrar sólo los datos seleccionados en un formulario; por ejemplo, los pedidos de un cliente especificado o los libros de un autor determinado. En este escenario, un usuario escribe información en un formulario y, a continuación, se ejecuta una consulta con la entrada del usuario como criterio; es decir, los datos se seleccionan basándose en una consulta parametrizada. La consulta sólo devuelve los datos que necesita el usuario.

El uso de consultas parametrizadas ayuda a la aplicación a ser más eficaz permitiendo a la base de datos realizar su trabajo en lo que es mejor, es decir, filtrando y ordenando registros. Al contrario, si solicita una tabla de base de datos completa, la transfiere en la red y, a continuación, utiliza lógica de la aplicación para buscar los registros que necesita, la aplicación podría llegar a ser lenta y difícil de utilizar.

En este tutorial, creará un formulario Windows Forms que muestra autores de la base de datos de ejemplo Pubs de SQL Server. Los usuarios podrán escribir un código de estados americanos (como CA para California) y llenar el conjunto de datos con una lista de autores que viven en ese estado.

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Pubs de SQL Server.

El tutorial está dividido en varias partes más pequeñas:

- Crear el formulario.
- Crear y configurar el conjunto de datos con respecto al cual se enlazará el formulario. Esto incluye crear una consulta que llene el conjunto de datos a partir de la base de datos.
- Agregar controles al formulario.
- Agregar código para actualizar el conjunto de datos con parámetros nuevos.
- Agregar código para desplazarse a lo largo de los autores.

## Crear el proyecto y el formulario

El primer paso es crear un formulario Windows Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, seleccione **Proyecto**.
2. Se muestra el cuadro de diálogo **Nuevo proyecto**.
3. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#** y, en el panel **Plantillas**, elija **Aplicación para Windows**.
4. Si ya tiene una solución abierta, seleccione **Cerrar solución**.

**Nota** En proyectos de producción, con mucha frecuencia tendrá varios proyectos en la misma solución. No obstante, en este tutorial, cerrará cualquier solución abierta y creará una nueva junto con el proyecto, por lo que no existirá ninguna interferencia entre lo que hará aquí y los formularios, conjuntos de datos, etc. existentes.

5. Asigne un nombre al proyecto que sea único y cumpla las convenciones de nomenclatura utilizadas. Por ejemplo, podría denominar este proyecto como **Walkthrough\_Parameters**. Cuando haya asignado un nombre, haga clic en **Aceptar** para crear el proyecto.

Visual Studio muestra un nuevo formulario en el Diseñador de Windows Forms.

## Crear y configurar un conjunto de datos

Como sucede con la mayoría de los escenarios de acceso a datos en Visual Studio, trabajará con un conjunto de datos. Un conjunto de datos es un contenedor, una caché, que contiene los registros con los que le interesa trabajar.

**Nota** Usar un conjunto de datos es sólo una de las opciones para obtener acceso a datos, y no es la elección ideal en algunos escenarios. No obstante, los conjuntos de datos son normalmente la elección correcta en las aplicaciones Windows Forms, por lo que usará uno de ellos en este tutorial. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos](#).

En este tutorial, agregará un conjunto de datos al formulario. No obstante, no lo hará directamente, sino manualmente, agregándolo al formulario. En su lugar, seguirá los pasos siguientes:

- Crear un adaptador de datos con un asistente. El adaptador contiene instrucciones SQL que se utilizan para leer y escribir información en la base de datos. El asistente le ayuda a definir las instrucciones SQL que necesita. Si es preciso, el asistente también crea una conexión a la base de datos.
- Generar el esquema del conjunto de datos. En este proceso, hará que Visual Studio cree una nueva clase conjunto de datos basándose en las tablas y columnas a las que está obteniendo acceso. Cuando genera la clase conjunto de datos, también agrega una instancia de ella al formulario.

Es importante que siga todos los procedimientos de esta sección. En caso contrario, el formulario no tendrá el conjunto de datos que se va a utilizar en las subsiguientes partes del tutorial.

Para obtener más información sobre adaptadores, vea [Introducción a los adaptadores de datos](#). Para obtener más información acerca de los conjuntos de datos, vea [Introducción a conjuntos de datos](#).

## Configurar una conexión de datos y un adaptador de datos

Para empezar, cree un adaptador de datos que contenga la instrucción SQL que se utilizará para llenar el conjunto de datos más adelante. Como parte de este proceso, defina una conexión para obtener acceso a la base de datos. Configure el adaptador de datos con el asistente, lo que facilita la creación de las instrucciones SQL necesarias para obtener acceso a los datos.

**Nota** Cuando el asistente haya finalizado, debe continuar en la sección siguiente para generar un conjunto de datos y completar la parte de acceso a datos del formulario.

### Para crear la conexión de datos y el adaptador de datos

1. Desde la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **OleDbDataAdapter** al formulario.

**Nota** También es posible usar el objeto **SqlDataAdapter**, que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa el objeto **OleDbDataAdapter** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB.

Se inicia el **Asistente para la configuración del adaptador de datos**, que ayuda a crear la conexión y el adaptador.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Pubs de SQL Server. Para llegar a la segunda página, haga clic en **Siguiente** en la primera página.
  - b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
  - c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT au_id, au_lname, state
FROM authors
WHERE (state = ?)
```

El signo de interrogación (?) es el marcador de posición del parámetro. (Si utiliza la clase **SqlDataAdapter** para tener acceso a SQL Server versión 7.0, los parámetros se especifican mediante variables con nombre.)

**Sugerencia** Para obtener ayuda sobre la generación de la instrucción SQL, haga clic en **Generador de consultas** para iniciar el **Generador de consultas**. Para proporcionar el marcador de posición del parámetro, escriba el signo de interrogación en la columna **Criterio** del campo de estado.

El asistente creará una conexión (**OleDbConnection1**) y un adaptador de datos (**OleDbDataAdapter1**).

3. Cuando el asistente haya finalizado, genere el conjunto de datos basándose en la consulta SQL que ha creado durante este procedimiento. Para obtener más información, vea la siguiente sección.

El adaptador de datos realmente contiene cuatro instrucciones SQL, no sólo la instrucción **Select** que ha creado, si no también instrucciones adicionales para actualizar, insertar y eliminar registros. En este tutorial, sólo se utilizará la instrucción **Select**.

La instrucción **Select** de SQL del adaptador se almacena como parte de la propiedad **SelectCommand** del adaptador de datos. La propiedad **SelectCommand** contiene un objeto de comando de datos (un objeto de tipo **OleDbDataComman**) que, a su vez, contiene no sólo la instrucción SQL, sino también otra información que necesita esa instrucción. Más concretamente, el comando contiene una colección **Parameters**. Esta colección contiene un objeto de parámetro (de tipo **OleDbParameter**) por cada parámetro que se va a pasar a y desde la instrucción SQL. Por supuesto, en este caso, sólo hay un parámetro, es decir, el valor de la columna de estado. De forma predeterminada, el Asistente para la configuración del adaptador de datos genera el parámetro con el nombre de la columna; por tanto, el parámetro se denomina "estado". Posteriormente, en este tutorial, aprenderá a establecer el valor del parámetro antes de ejecutar la instrucción SQL.

## Crear el conjunto de datos

Después de haber establecido los métodos para conectarse a la base de datos y de especificar la información que desea (a través del comando SQL del adaptador de datos), puede hacer que Visual Studio cree un conjunto de datos. Visual Studio puede generar el conjunto de datos automáticamente basándose en la consulta que ha especificado para el adaptador de datos. El conjunto de datos es una instancia de la clase **DataSet** basada en un esquema XML correspondiente (archivo .xsd) que describe los elementos de la clase (tabla, columnas y restricciones). Para obtener más información acerca de la relación entre los conjuntos de datos y los esquemas, vea [Introducción al acceso a datos con ADO.NET](#).

### Para generar un conjunto de datos

1. Desde el menú **Datos**, elija **Generar conjunto de datos**.

**Sugerencia** Si no ve el menú **Datos**, haga clic en el formulario; el formulario debe tener el foco para que aparezca el menú.

Se mostrará el cuadro de diálogo **Generar conjunto de datos**.

2. Póngale por nombre **dsAuthors** al conjunto de datos y active la casilla **Agregar este conjunto de datos al diseñador**; a continuación, haga clic en **Aceptar**.

Visual Studio genera una clase de conjunto de datos con tipo (**dsAuthors**) y un esquema que define el conjunto de datos. Verá el nuevo esquema (**dsAuthors.xsd**) en el **Explorador de soluciones**. Si elige la opción **Mostrar todos los archivos** del **Explorador de soluciones**, verá que el archivo de esquema tiene un archivo .jsl dependiente que contiene el código que define la nueva clase del conjunto de datos.

Por último, Visual Studio agrega una instancia de la nueva clase de conjunto de datos (**dsAuthors1**) al formulario.

Llegados a este punto, tiene configurado todo lo necesario para obtener información de la base de datos y almacenarla en un conjunto de datos. Ya puede crear un formulario que muestre los datos.

## Agregar controles para mostrar datos

Para este tutorial, el formulario necesitará una manera en la que los usuarios escriban un código de estado, un modo de ejecutar la consulta y otro para mostrar información del autor.

### Para agregar controles al formulario

1. Agregue los siguientes controles al formulario, y deles un nombre como se indica:

Control	Propósito	Nombre	Texto
<b>TextBox</b>	Permite que los usuarios escriban el código de estado de los autores que desean ver	<b>txtStateParameter</b>	(Cadena vacía)
<b>Button</b>	Ejecuta la consulta y llena el conjunto de datos	<b>btnShow</b>	Show
<b>TextBox</b>	Muestra el identificador del autor	<b>txtAuthorID</b>	(Cadena vacía)
<b>TextBox</b>	Muestra el apellido del autor	<b>txtAuthorLName</b>	(Cadena vacía)
<b>TextBox</b>	Muestra el estado donde vive el autor (se muestra en el tutorial sólo como ejemplo ilustrativo).	<b>txtAuthorState</b>	(Cadena vacía)

2. Agregue etiquetas delante de los cuadros de texto para indicar su función.

## Agregar código para llenar el conjunto de datos

Cuando se muestra el formulario, el conjunto de datos no se llena automáticamente con información de la base de datos. Si no se debe llenar el conjunto de datos explícitamente. En este tutorial, cada vez que el usuario hace clic en el botón **Mostrar**, se vuelve a ejecutar la consulta especificada en el adaptador de datos y el resultado alimenta el conjunto de datos. En este caso, la consulta requiere un parámetro. El valor del parámetro es la información que el usuario escribe en el cuadro de texto **txtStateParameter**.

Ejecute la consulta en código llamando al método **Fill** del adaptador de datos.

### Para escribir código para llenar el conjunto de datos

- Haga doble clic en el botón **Mostrar** para crear un método para el evento **Click**. Agregue código al controlador para:
  - Establecer el valor del único parámetro requerido por la instrucción SQL que ha creado antes. Lo establecerá en el valor que el usuario haya escrito en el cuadro de texto **txtStateParameter**.
  - Llamar al método **Clear** del conjunto de datos. Si no lo borra, los registros devueltos por una consulta se anexan al conjunto de datos.
  - Llamar al método **Fill** del adaptador de datos, pasándole una referencia al conjunto de datos y al valor del parámetro que se va a incluir en la consulta.

En el ejemplo siguiente se muestra la apariencia del código para el método:

```
// Visual J#
private void btnShow_Click(Object sender, System.EventArgs e)
{
    OleDbDataAdapter1.get_SelectCommand().get_Parameters().get_Item("state").set_Value ( txtStateParameter.get_Text());
    dsAuthors1.Clear();
    OleDbDataAdapter1.Fill(dsAuthors1);
    OleDbConnection1.Close();
}
```

## Enlazar los cuadros de texto al conjunto de datos

Los tres cuadros de texto de autores pretenden mostrar información de un único registro en la tabla de datos Authors del conjunto de datos. Para hacer que esto se produzca automáticamente, enlace columnas de la tabla de datos a cuadros de texto. El mecanismo de enlace del formulario garantiza que los controles del cuadro de texto se actualizan automáticamente cada vez que se mueve a otro registro.

En el tutorial, enlazará la propiedad **Text** de los controles del cuadro de texto a la columna que desee mostrar. En realidad, puede enlazar cualquier propiedad del control de cuadro de texto a cualquier columna del conjunto de datos y también puede enlazar varias propiedades de un control. Para obtener más información, vea [Arquitectura de datos en formularios Windows Forms](#).

### Para enlazar cuadros de texto al conjunto de datos

1. Vuelva al diseñador de formularios.
2. Seleccione el cuadro de texto del primer autor y presione F4 para abrir la ventana Propiedades.
3. Expanda el nodo **(DataBindings)** y, para la propiedad **Text**, expanda **dsAuthors**, expanda **authors**, y seleccione **au\_id** de la lista desplegable.
4. Repita los pasos 2 y 3 para los otros dos cuadros de texto de autores restantes, enlázelos a **dsAduthors1.authors.au\_lname** y **dsAduthors1.authors.state**, respectivamente.

Ahora, los controles están en disposición de mostrar datos de cada fila del conjunto de datos. No obstante, aún necesita una forma para moverse de una fila a la siguiente.

## Agregar controles de desplazamiento

Finalmente, agregue controles de desplazamiento al formulario. En este tutorial, agregará los botones Anterior y Siguiente. (También podría agregar los botones Primero y Último, pero sólo son sencillas variaciones de Anterior y Siguiente.) También agregará un cuadro de texto que muestra la posición del registro actual.

En un formulario Windows Forms, la información acerca de la posición actual y el contador de registros en una tabla de datos

individual, así como la información acerca de los enlaces entre tablas de datos y controles, se mantiene en un objeto **CurrencyManager**. En un formulario pueden existir varias tablas de datos (todas con enlaces distintos y contadores y posiciones en tiempo de ejecución), cada una con su propio objeto **CurrencyManager**. Para administrar los distintos objetos **CurrencyManager** posibles, el formulario incluye un objeto **BindingContext** que proporciona una interfaz única para todos los administradores de listas.

La posición del registro actual en una tabla de datos está disponible en la propiedad **Position** que se expone a través del objeto **BindingContext**. Para desplazarse, cambie el valor de esta propiedad. Para determinar el número de registros que hay en una tabla, puede consultar la propiedad **Count** del objeto **BindingContext**.

### Para agregar controles de desplazamiento

1. Agregue los controles siguientes al formulario y establezca las propiedades como se indica:

Control	Nombre	Texto
Button	btnNext	Siguiente
Button	btnPrevious	Anterior

2. Cree un método de control de eventos para el evento **Click** del botón **Anterior**. Agregue código para disminuir la propiedad **Position** del objeto **BindingContext**. El código será como el que aparece a continuación:

```
// Visual J#
private void btnPrevious_Click(Object sender, System.EventArgs e)
{
    this.get_BindingContext().get_Item(dsAuthors1, "authors").set_Position (this.get_BindingContext().get_Item(dsAuthors1, "authors").get_Position () -1 ) ;
}
```

3. Haga lo mismo para el botón **Siguiente**, sólo aumente la posición. Utilice código como el siguiente:

```
// Visual J#
private void btnNext_Click(Object sender, System.EventArgs e)
{
    this.get_BindingContext().get_Item(dsAuthors1, "authors").set_Position (this.get_BindingContext().get_Item(dsAuthors1, "authors").get_Position () + 1 ) ;
}
```

### Mostrar la posición del registro actual

Finalmente, puede crear un método **ShowPosition** para mostrar la posición del registro actual.

#### Para mostrar la posición del registro actual

1. Agregue un control **TextBox** al formulario y póngale el nombre **txtPosition**.

**Sugerencia** Establezca la propiedad **Enabled** en **False**, así esos usuarios pueden ver la posición actual, pero no pueden cambiar el valor del control.

2. Cree un método en el formulario denominado **ShowPosition**. En el método, obtenga la posición actual del objeto **BindingContext** y muéstrela en el cuadro de texto. En el ejemplo siguiente se muestra la apariencia del código:

```
// Visual J#
private void ShowPosition()
{
    int iCnt;
    int iPos;
    iCnt = this.get_BindingContext().get_Item(dsAuthors1, "authors").get_Count();
    iPos = this.get_BindingContext().get_Item(dsAuthors1, "authors").get_Position () + 1;
    if(iCnt == 0)
    {
        txtPosition.set_Text ("(No records)");
    }
    else
```

```

    {
        txtPosition.set_Text( iPos + " of " + iCnt );
    }
}

```

3. Agregue una llamada al método **ShowPosition** en cualquier lugar del código donde pueda cambiar la posición registro actual. Para este tutorial, agréguelo en estos lugares:

- Después de llamar al método **Fill** en el método de control de eventos **Click** para el botón **Mostrar**.
- Después de cambiar la posición del registro en los métodos **Click** para los botones **Anterior** y **Siguiente**. Por ejemplo, el método de control de eventos completo para el método **Previous** presentará la apariencia siguiente:

```

// Visual J#
private void btnPrevious_Click(Object sender, System.EventArgs e)
{
    this.get_BindingContext().get_Item(dsAuthors1, "authors").set_Position (this.get_
BindingContext().get_Item(dsAuthors1, "authors").get_Position () - 1 );

    ShowPosition();
}

```

## Pruebas

Ahora puede comprobar el formulario para asegurarse de que muestra los datos correctamente basándose en el parámetro que escriba.

### Para comprobar el formulario

1. Presione F5 para ejecutar el formulario.
2. Cuando se muestre el formulario, escriba **CA** en el cuadro de texto de estado y, a continuación, haga clic en **Mostrar**.  
Se muestra el primer autor que vive en California.
3. Haga clic en los botones **Anterior** y **Siguiente** para desplazarse a través de los autores.
4. Escriba un nuevo valor en el cuadro de texto de estado (por ejemplo, pruebe con **UT** para Utah) y, a continuación, haga clic en **Mostrar**.

Confirme que se muestra un autor nuevo y que también ha cambiado el contador de registros.

## Pasos siguientes

Cuando haya terminado este tutorial, tendrá un formulario enlazado a datos sencillo. Hay varias mejoras que podría hacer, entre ellas:

- Mostrar códigos de estado en un cuadro de lista desplegable en vez de hacer que los usuarios los escriban. Una forma podría ser agregar otro adaptador de datos al formulario con la instrucción SQL `SELECT DISTINCT state FROM authors`, generar un segundo conjunto de datos y enlazar un cuadro de lista a ese conjunto de datos.
- Comprobar que la consulta ha devuelto registros. Para ello, podría comprobar la propiedad **Count** del objeto **BindingContext**; si es cero, puede mostrar un mensaje. En el ejemplo siguiente se muestra una versión expandida del controlador del botón **Mostrar** que incorpora esta comprobación:

```

// Visual J#
private void btnShow_Click(Object sender, System.EventArgs e)
{
    OleDbDataAdapter1.get_SelectCommand().get_Parameters().get_Item("state").set_Value ( t
xtStateParameter.get_Text());
    dsAuthors1.Clear();
    OleDbDataAdapter1.Fill(dsAuthors1);
    if(this.get_BindingContext().get_Item(dsAuthors1, "authors").get_Count() == 0)
    {
        MessageBox.Show("No authors found in " + txtStateParameter.get_Text());
        txtStateParameter.Focus();
    }
}

```

```
    }  
    else  
    {  
        ShowPosition();  
    }  
}
```

- Mostrar los autores de una cuadrícula enlazada a datos en vez de controles individuales. De este modo, podría ver a todos los autores simultáneamente.
- Agregar botones que permitan al usuario agregar, modificar y eliminar autores.

### **Vea también**

[Introducción a conjuntos de datos](#) | [Introducción a los adaptadores de datos](#) |  
[Arquitectura de datos en formularios Windows Forms](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: Llamar a los servicios Web XML desde los formularios Windows Forms con Visual J#

Un nuevo aspecto de Visual Studio lo forman los servicios Web XML, que proporcionan la capacidad de intercambiar mensajes en un entorno débilmente acoplado mediante protocolos estándar como HTTP, XML, XSD, SOAP y WSDL. Los mensajes pueden estructurarse y dotarse de tipos o estar débilmente definidos. Debido a que las aplicaciones de servicios Web se basan en protocolos estándar, pueden comunicarse con una amplia gama de implementaciones, plataformas y dispositivos. Para obtener más información, vea [Servicios Web XML en código administrado](#).

Los servicios Web pueden usarse para mejorar las funciones de los formularios Windows Forms. Conectar formularios Windows Forms con servicios Web es tan sencillo como realizar llamadas a los métodos de servicios Web, los cuales se procesan en un servidor que a continuación devuelve los resultados de la llamada al método.

Hay dos tipos de métodos de servicio Web, síncrono y asíncrono. Cuando se llama a métodos síncronos de servicios Web, el código de llamada espera a que el servicio Web responda antes de continuar sus operaciones. Cuando se llama a métodos asíncronos de servicios Web, se puede continuar utilizando el subproceso de llamada mientras se espera a que responda el servicio Web. Ello permite utilizar el conjunto existente de subprocesos de manera eficiente en la aplicación cliente. Para obtener más información sobre el trabajo con métodos síncronos y asíncronos de servicios Web, vea [Obtener acceso a servicios Web en código administrado](#).

## Métodos síncronos de servicios Web

Una llamada a un método síncrono de servicio Web implica llamar al método y esperar a que se produzca el cálculo en el servidor y devuelva un valor, antes de continuar con el resto del código del formulario Windows Forms.

### Para crear un servicio Web XML

1. Cree una aplicación de servicio Web. Para obtener más información, vea [Crear servicios Web XML en código administrado](#). Acepte el nombre predeterminado de `WebService1`.
2. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el archivo `.asmx` y elija **Ver código**.
3. Cree un método de servicio Web que realice sumas. Este método de servicio Web toma dos números enteros y los suma, devolviendo el valor resultante:

```
// Visual J#
/** @attribute WebMethod() */
public int WebAdd(int x, int y)
{
    return x + y;
}
```

4. Cree un método de servicio Web que realice multiplicaciones. Este método de servicio Web toma dos números enteros y los multiplica, devolviendo el producto:

```
// Visual J#
/** @attribute WebMethod() */
public int WebMultiply(int x, int y)
{
    return x * y;
}
```

5. En el menú **Generar**, elija **Generar solución**. También puede desplazarse al archivo `.asmx` que creó en el proyecto para aprender más sobre los servicios Web. El servicio Web está ahora disponible para llamarlo desde un formulario Windows Forms.

### Para llamar a un servicio Web XML síncronamente

1. Cree una nueva aplicación Windows. Para obtener más información, vea [Crear un proyecto de aplicación para Windows](#).
2. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo References del proyecto y, a continuación, haga clic en **Agregar referencia Web**.

Se abrirá el cuadro de diálogo **Agregar referencia Web**.



3. Escriba el siguiente identificador URI de servicio Web en el cuadro **Dirección** y presione ENTRAR:

http://localhost/WebService1/Service1.asmx

Éste es el identificador URI del servicio Web que creó en el procedimiento anterior. Los métodos Web **WebAdd** y **WebMultiply** aparecerán en el panel izquierdo del cuadro de diálogo **Agregar referencia Web**.

4. Haga clic en **Agregar referencia**.
5. En el Cuadro de herramientas, agregue tres controles **TextBox** y dos controles **Button**. Los cuadros de texto serán para los números y los botones se emplearán para los cálculos y para llamar a los métodos de servicio Web.
6. Establezca las propiedades de estos controles de la manera siguiente:

Control	Propiedad	Texto
TextBox1	Text	0
TextBox2	Text	0
TextBox3	Text	0
Button1	Text	Sumar
Button2	Text	Multiplicar

7. Haga clic con el botón secundario del *mouse* y elija **Ver código**.
8. Cree una instancia del servicio Web como miembro de la clase. Necesitará conocer el nombre del servidor donde creó el servicio Web anterior.

```
// Visual J#
localhost.Service1 MathServiceClass = new localhost.Service1();
```

9. Cree un controlador para el evento **Click** del control Button1. Para obtener información detallada, vea [Crear controladores de eventos en el Diseñador de Windows Forms](#).

```
// Visual J#
private void button1_Click (Object sender, System.EventArgs e)
{
    // Create instances of the operands and result.
    int x, y, z;
    // Parse the contents of the text boxes into integers.
    x = Integer.parseInt(textBox1.get_Text());
    y = Integer.parseInt(textBox2.get_Text());
    // Call the WebAdd Web service method from the instance of the Web service.
    z = MathServiceClass.WebAdd(x, y);
    textBox3.set_Text(""+z);
}
```

10. Cree un controlador de eventos para el evento **Click** de Button2 de la misma forma, y agregue el código siguiente.

```
// Visual J#
private void button2_Click (Object sender, System.EventArgs e)
{
    // Create instances of the operands and result.
    int x, y, z;
    // Parse the contents of the text boxes into integers.
    x = Integer.parseInt(textBox1.get_Text());
    y = Integer.parseInt(textBox2.get_Text());
    // Call the WebAdd Web service method from the instance of the Web service.
    z = MathServiceClass.WebMultiply(x, y);
    textBox3.set_Text(""+z);
}
```

11. Presione F5 para ejecutar la aplicación. Escriba los valores de los dos primeros cuadros de texto. Al presionar el botón **Sumar**, el tercer cuadro de texto debe mostrar la suma. Al presionar el botón **Multiplicar**, el tercer cuadro de texto debe mostrar el producto.

**Nota** La primera llamada a un servicio Web lleva un tiempo para que el servidor lo procese, ya que su instancia reside en el servidor. Tenga este punto en cuenta al presionar los botones de la aplicación. Este retardo se trata en la sección siguiente.

## Servicios Web asincrónicos

Al llamar a métodos de servicios Web asincrónicos, la aplicación continúa ejecutándose mientras espera a que el servicio Web responda. Ello permite utilizar los recursos eficientemente en la aplicación cliente. Se trata de una forma mucho más eficaz en cuanto a recursos de implementar servicios Web en una aplicación Windows.

Para obtener información detallada, vea [Obtener acceso a un servicio Web de manera asincrónica en código administrado](#).

### Vea también

[Obtener acceso a servicios Web XML en código administrado](#) | [Servicios Web XML en código administrado](#) | [Recomendaciones para formularios Windows Forms y Web Forms](#) | [Referencias Web](#) | [Generar un proxy de servicio Web XML](#) | [Aplicaciones para Windows como clientes](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutorial: crear una aplicación de servicios de Windows en el Diseñador de componentes con Visual J#

Los procedimientos de este tema conforman el proceso de creación de una sencilla aplicación de servicios de Windows que escribe mensajes en un registro de eventos. Los pasos básicos que se realizan para crear y utilizar el servicio son los siguientes:

- Crear un proyecto mediante la plantilla de aplicación de servicios de Windows. Esta plantilla crea una clase que hereda de **ServiceBase** y escribe gran parte del código básico del servicio, como es el código que inicia el servicio.
- Escribir el código para los procedimientos **OnStart** y **OnStop**, y reemplazar los otros métodos que se quiera volver a definir.
- Agregar los instaladores necesarios para su aplicación de servicio. De forma predeterminada, al hacer clic en el vínculo **Agregar instalador**, se agrega a la aplicación una clase que contiene dos o más instaladores: uno para instalar el proceso y otro para cada uno de los servicios asociados que contiene el proyecto.
- Generar el proyecto.
- Crear un proyecto de instalación para instalar el servicio y, a continuación, instalarlo.
- Abrir el Administrador de control de servicios de Windows 2000 e iniciar el servicio.

Para empezar, debe crear el proyecto y definir los valores necesarios para que el servicio funcione correctamente.

## Para crear y configurar el servicio

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. Seleccione el proyecto **Servicio de Windows** de la lista de plantillas de proyecto de Visual J# y asígnele el nombre **MyNewService**.

**Nota** La plantilla de proyecto agrega automáticamente una clase componente, denominada Service1, que hereda de **System.ServiceProcess.ServiceBase**.

3. Haga clic en el diseñador. A continuación, en la ventana Propiedades, asigne a la propiedad **ServiceName** de Service1 el valor **MyNewService**.
4. Asigne a la propiedad **Name** el valor **MyNewService**.
5. Asigne a la propiedad **AutoLog** el valor **true**.
6. En el Editor de código, edite el método **Main** para crear una instancia de **MyNewService**. Cuando cambió el nombre del servicio en el paso 4, el nombre de la clase no se modificó en el método **Main**.

```
// Visual J#
public static void main(String[] args)
{
    System.ServiceProcess.ServiceBase[] ServicesToRun;
    ServicesToRun = new System.ServiceProcess.ServiceBase[]
        { new MyNewService() };
    System.ServiceProcess.ServiceBase.Run(ServicesToRun);
}
```

En la siguiente sección, agregará un registro de eventos personalizado al servicio de Windows. Los registros de eventos no están asociados de ningún modo a los servicios de Windows. Aquí, el componente **EventLog** se utiliza como ejemplo del tipo de componente que se puede agregar a un servicio de Windows. Para obtener más información sobre registros de eventos personalizados, vea [Crear y quitar registros de eventos personalizados](#).

## Para agregar la funcionalidad de registros de eventos personalizado al servicio

1. En el Explorador de soluciones, haga clic con el botón secundario en Service1.jsl y seleccione Diseñador de vistas.
2. En la ficha **Componentes** del Cuadro de herramientas, arrastre un componente **EventLog** hasta el diseñador.
3. En el Explorador de soluciones, haga clic con el botón secundario en Service1.jsl y seleccione Ver código.
4. Modifique el constructor para definir un registro de eventos personalizado:

```
// Visual J#
```

```
public MyNewService()
{
    InitializeComponent();
    if (!System.Diagnostics.EventLog.SourceExists("MySource")) {
        System.Diagnostics.EventLog.CreateEventSource(
            "MySource", "MyNewLog");
    }
    eventLog1.set_Source("MySource");
    eventLog1.set_Log("MyNewLog");
}
```

### Para definir qué debe ocurrir al iniciar el servicio

- En el Editor de código, busque el método **OnStart**, que fue reemplazado automáticamente al crear el proyecto, y escriba código que determine las acciones que se deben realizar al empezar a ejecutar el servicio:

```
// Visual J#
protected void OnStart(String[] args)
{
    eventLog1.WriteEntry("In OnStart");
}
```

**Nota** Una aplicación de tipo servicio está diseñada para ejecutarse de forma prolongada. Por ello, normalmente sondea o supervisa algún elemento del sistema. Esa supervisión se puede establecer en el método **OnStart**. No obstante, **OnStart** no realiza realmente la supervisión. El método **OnStart** debe volver al sistema operativo una vez que el funcionamiento del servicio ha empezado. No debe bloquearse ni ejecutar un bucle infinito. Para establecer un mecanismo de sondeo sencillo, puede utilizar el componente **System.Timers.Timer**. En el método **OnStart**, definiría los parámetros del componente y, a continuación, asignaría a la propiedad **Timer.Enabled** el valor **true**. El temporizador activaría entonces los eventos periódicamente en el código y, en esos instantes, el servicio podría realizar su control.

### Para definir qué debe ocurrir al detener el servicio

- En el Editor de código, busque el método **OnStop**, que fue reemplazado automáticamente al crear el proyecto, y escriba código que determine las acciones que se deben realizar cuando se detiene el servicio:

```
// Visual J#
protected void OnStop()
{
    eventLog1.WriteEntry("In onStop.");
}
```

También puede reemplazar los métodos **OnPause**, **OnContinue** y **OnShutdown** para definir un procesamiento adicional para el componente.

### Para definir otras acciones para el servicio

- Para el método que desee controlar, reemplace el método apropiado y defina lo que desea que suceda.

El código siguiente muestra el aspecto del método **OnContinue** al ser reemplazado:

```
// Visual J#
protected void OnContinue()
{
    eventLog1.WriteEntry("In OnContinue.");
}
```

Cuando se instala un servicio de Windows, se deben ejecutar algunas acciones personalizadas; esto se consigue mediante la clase **Installer**. Visual Studio puede crear estos instaladores específicamente para un servicio de Windows y agregarlos al proyecto.

## Para crear los instaladores necesarios para el servicio

1. Vuelva a la vista Diseño para Service1.
2. Haga clic en el fondo del diseñador para seleccionar el propio servicio, en vez de cualquier elemento de su contenido.
3. En la ventana Propiedades, haga clic en el vínculo **Agregar instalador**, situado en el área gris, bajo la lista de propiedades.

De forma predeterminada, se agrega al proyecto una clase de componente que contiene dos instaladores. El componente se denomina **ProjectInstaller**, y los instaladores que contiene son el instalador para el servicio y el instalador para el proceso asociado al servicio.

4. Utilice la vista Diseño para **ProjectInstaller** y haga clic en **ServiceInstaller1**.
5. En la ventana Propiedades, asigne a la propiedad **ServiceName** el valor **MyNewService** (si es necesario).
6. Asigne a la propiedad **StartType** el valor **Automatic**.

## Para generar el proyecto de servicio

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) y elija **Propiedades** en el menú contextual. Aparece el cuadro de diálogo **Páginas de propiedades**.
2. En el panel izquierdo, seleccione la ficha **General** en la carpeta **Propiedades comunes**.
3. En la lista **Objeto inicial**, elija **MyNewService.MyNewService**. Haga clic en **Aceptar**.
4. Presione Ctrl+Mayús+B para generar el proyecto.

Ahora que el proyecto se ha generado, puede ser implantado en el sistema. Un proyecto de instalación instalará los archivos del proyecto compilados y ejecutará los instaladores necesarios para ejecutar el servicio de Windows. Para crear un proyecto de instalación completo, deberá agregar el resultado del proyecto (**MyNewService.exe**) al proyecto de instalación y, a continuación, agregar una acción personalizada para instalar **MyNewService.exe**. Para obtener más información sobre los proyectos de instalación, vea [Proyectos de instalación](#). Para obtener más información sobre acciones personalizadas, vea [Tutorial: crear una acción personalizada](#).

## Para crear un proyecto de instalación para el servicio

1. En el menú **Archivo**, seleccione **Agregar proyecto** y, a continuación, haga clic en **Nuevo proyecto**.
2. En el panel **Tipos de proyecto**, seleccione la carpeta **Proyectos de instalación e implementación**.
3. En el panel **Plantillas**, seleccione **Proyecto de instalación**. Asigne al proyecto el nombre **MyServiceSetup**.

Se agregará un proyecto de instalación a la solución. A continuación, deberá agregar el resultado del proyecto de servicios de Windows (**MyNewService.exe**) a la instalación.

## Para agregar MyNewService.exe al proyecto de instalación

1. En el Explorador de soluciones, haga clic con el botón secundario en **MyServiceSetup**, elija **Agregar** y, a continuación, **Resultados del proyecto**.

Aparecerá el cuadro de diálogo **Agregar grupo de resultados del proyecto**.

2. **MyNewService** aparece seleccionado en el cuadro **Proyecto**.
3. En el cuadro de lista, seleccione **Resultado principal** y haga clic en **Aceptar**.

Se agrega al proyecto de instalación un elemento del proyecto para el resultado principal de **MyNewService**. Ahora, agregue una acción personalizada para instalar el archivo MyNewService.exe.

## Para agregar una acción personalizada al proyecto de instalación

1. En el Explorador de soluciones, haga clic con el botón secundario en el proyecto de instalación, elija **Ver** y, a continuación, **Acciones personalizadas**.

Aparece el editor de **Acciones personalizadas**.

2. En el editor de **Acciones personalizadas**, haga clic con el botón secundario en el nodo **Acciones personalizadas** y elija **Agregar acción personalizada**.

Aparecerá el cuadro de diálogo **Seleccionar elemento en el proyecto**.

3. Haga doble clic en la opción **Carpeta de la aplicación** del cuadro de lista, para abrirla, seleccione **Resultado principal de MyNewService (Activo)** y haga clic en **Aceptar**.

El resultado principal se agrega a los cuatro nodos de las acciones personalizadas: **Instalar**, **Confirmar**, **Deshacer** y **Desinstalar**.

4. Genere el proyecto de instalación.

### Para instalar el servicio de Windows

- Sitúese en el directorio donde se almacenó el proyecto de instalación y ejecute el archivo .msi para instalar **MyNewService.exe**.

**Nota** Es posible que deba especificar un dominio y un nombre de usuario (por ejemplo, `MyDomain\MyAlias`) en el cuadro de diálogo **Establecer inicio de sesión del servicio**.

### Para iniciar y detener el servicio

1. Abra el Administrador de control de servicios mediante uno de los métodos siguientes:
  - En Windows 2000 Professional, haga clic con el botón secundario en el icono **Mi PC** del escritorio y, a continuación, haga clic en **Administrar**. En el cuadro de diálogo **Administración de equipos**, expanda el nodo **Servicios y Aplicaciones**.  
O bien
  - En Windows 2000 Server, haga clic en **Inicio**, elija **Programas**, después, **Herramientas administrativas** y, a continuación, haga clic en **Servicios**.

**Nota** En Windows NT versión 4.0, puede abrir este cuadro de diálogo desde el Panel de control.

Podrá ver el servicio **MyNewService** en la lista de la sección **Servicios** de la ventana.

2. Seleccione su servicio en la lista, haga clic en él con el botón secundario y luego haga clic en **Iniciar**.
3. Haga clic en el servicio con el botón secundario y, a continuación, haga clic en **Detener**.

### Para comprobar el registro de eventos del servicio

1. Vaya al Explorador de servidores y busque el nodo **Registros de eventos**. Para obtener más información, vea [Trabajar con registros de eventos en el Explorador de servidores](#).
2. Busque la entrada correspondiente a **MyNewLog** y expándala. Aparecerán las entradas correspondientes a las acciones realizadas por el servicio.

### Para desinstalar el servicio

- En el menú **Inicio**, abra **Panel de control**, haga clic en **Agregar o quitar programas**, seleccione el servicio y haga clic en **Desinstalar**.
- También puede desinstalar el programa haciendo clic con el botón secundario en el icono de programa correspondiente al archivo .msi y seleccionando **Desinstalar**.

### Pasos siguientes

Podría probar también el uso de un componente **ServiceController** para enviar comandos al servicio que ha instalado. Para obtener más información sobre el uso del componente **ServiceController**, vea [Supervisar servicios de Windows](#).

### Vea también

[Aplicaciones de servicios de Windows](#) | [Introducción a las aplicaciones de servicios de Windows](#) | [Agregar instaladores a una aplicación de servicio](#) | [Instalar y desinstalar servicios](#) | [Depurar aplicaciones de servicios de Windows](#) | [Iniciar el Visor de eventos](#) | [Acceso e inicialización del Explorador de servidores](#) | [Tutoriales sobre los formularios Windows Forms](#)

# Tutoriales sobre los formularios Web Forms

Los siguientes tutoriales le orientarán en el uso de páginas de formularios Web Forms y de ASP.NET.

## En esta sección

### [Crear una página básica de formularios Web Forms con Visual J#](#)

Muestra las técnicas básicas para crear una página de formularios Web Forms, agregarle controles y ejecutarla.

### [Crear una aplicación Web utilizando Visual J#](#)

Explica cómo se escribe una aplicación de formularios Web Forms para, después, integrar un componente de objeto comercial que aplique un descuento del 10% en compras.

### [Crear una aplicación Web utilizando un objeto comercial de otro fabricante con Visual J#](#)

Explica cómo se escribe una aplicación de formularios Web Forms utilizando extensiones administradas de C++ para, después, integrar un componente de objeto comercial de otro fabricante que aplique un descuento del 10 % en compras.

### [Validar los datos proporcionados por el usuario en una página de formularios Web Forms con Visual J#](#)

Explica cómo utilizar controles de validación de formularios Web Forms para comprobar entradas de usuario sin código (por ejemplo, entradas requeridas, tipos de datos, patrones y valores específicos).

### [Mostrar datos en una página de formularios Web Forms con Visual J#](#)

Muestra un escenario de acceso a datos básico, con instrucciones paso a paso para crear una página de formularios Web Forms que muestre datos en un control de cuadrícula.

### [Crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J#](#)

Describe cómo usar un comando de datos y un lector de datos de ADO.NET para proporcionar acceso optimizado a datos de sólo lectura en una página de formularios Web Forms.

### [Actualizar datos mediante una consulta de actualización de bases de datos en los formularios Web Forms con Visual J#](#)

Describe las técnicas básicas para crear una página de formularios Web Forms que use un comando de datos para actualizar la base de datos.

### [Utilizar un control Web DataGrid para leer y escribir datos con Visual J#](#)

Proporciona instrucciones paso a paso sobre cómo utilizar un control de cuadrícula para permitir que los usuarios vean y editen datos.

### [Crear acceso a datos paginados mediante una página de formularios Web Forms con Visual J#](#)

Explica cómo crear una página de formularios Web Forms que permita a los usuarios ver registros de datos (unos pocos registros cada vez) avanzando y retrocediendo para ver los registros posteriores y anteriores.

### [Crear un control Web de usuario con Visual J#](#)

Explica cómo crear un control de usuario de formularios Web Forms (es decir, una página configurada de tal modo que actúe como un control) y utilizarlo en otra página.

### [Convertir una página de formularios Web Forms en un control de usuario con Visual J#](#)

Explica cómo se convierte una página básica de formularios Web Forms en un control de usuario que puede alojarse en otra página.

### [Crear un control Web personalizado con Visual J#](#)

Explica cómo crear un control personalizado, agregarlo al cuadro de herramientas y utilizarlo en una página de formularios Web Forms.

### [Mostrar un documento XML en una página de formularios Web Forms mediante transformaciones con Visual J#](#)

Proporciona información detallada sobre cómo leer un archivo XML y mostrarlo de diferentes maneras en una página de formularios Web Forms.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, trabajar con formularios Windows Forms y Web Forms, y realizar tareas relacionadas con datos.

### [Páginas de formularios Web Forms](#)

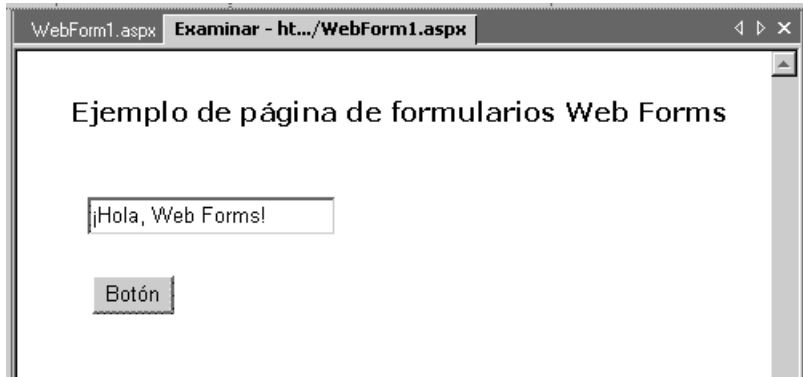
Tema de Visual Basic/Visual C# que ofrece vínculos a información acerca de las tecnologías y herramientas para crear la interfaz de usuario basada en explorador para aplicaciones Web ASP.NET.

# Tutorial: crear una página básica de formularios Web Forms con Visual J#

Este tutorial muestra cómo crear una página de formularios Web Forms. Se muestra cómo agregar controles a la página y a crear código para aquellos que se ejecutarán en el servidor. Las tareas ilustradas en este tutorial incluyen:

- Crear una nueva página de formularios Web Forms.
- Agregar controles y texto HTML estático a la página.
- Crear un controlador de eventos para el control.
- Generar y ejecutar la página de formularios Web Forms.

Cuando termine, la página de formularios Web Forms tendrá el siguiente aspecto:



## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo `http://`, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

**Sugerencia** Si ya tiene abierta una solución, seleccione **Cerrar solución** para que el nuevo proyecto de formularios Web Forms forme parte de su propia solución.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada `WebForm1.aspx` en la vista Diseño del Diseñador de Web Forms.

## Examinar la estructura de formularios Web Forms

Dedique un momento a observar cómo están estructuradas las páginas de formularios Web Forms y cómo se coloca el Diseñador de Web Forms. El Diseñador de Web Forms se abre con un archivo denominado `WebForm1.aspx`. Una página de formularios Web Forms está formada por dos archivos independientes:

- El archivo `.aspx` contiene el texto HTML y los controles que constituyen la interfaz de usuario de la página.
- Un archivo separado, denominado `WebForm1.aspx.jsl`, contiene el código de la página, es decir, es el archivo de clase de la página. También se denomina archivo de código subyacente. De manera predeterminada, el Explorador de soluciones no muestra el archivo de clase de la página.

### Para ver el archivo de clase de una página en el Explorador de soluciones

- Haga clic en el botón **Mostrar todos los archivos** en la barra de herramientas del Explorador de soluciones y, a continuación, expanda el nodo de `WebForm1.aspx`.



En la parte inferior del Diseñador de Web Forms existen dos fichas, **Diseño** y **HTML**, que muestran diferentes vistas del archivo .aspx con el que está trabajando:

- La vista **Diseño** le proporciona una vista WYSIWYG donde puede arrastrar controles y utilizar la ventana **Propiedades** para configurarlos.
- La vista **HTML** le muestra la misma información, pero "sin formato", como ocurre con los archivos HTML. Como en los archivos HTML, el Diseñador de Web Forms admite IntelliSense para los elementos en la vista HTML.

Puede trabajar en cualquiera de las dos vistas. Cuando cambia entre las dos vistas, cada una de ellas se actualiza con los cambios efectuados en la otra. Como verá más adelante, se utiliza el editor de código para escribir código para el archivo de clase de la página (.aspx.jsl).

## Agregar controles y texto

Ahora puede agregar controles y texto a la página.

### Modo Cuadrícula o modo Flujo

De manera predeterminada, la página de formularios Web Forms está en modo cuadrícula. En este modo, puede arrastrar controles a la página y colocarlos utilizando sus coordenadas absolutas (x e y). En exploradores de bajo nivel, los controles se colocan mediante tablas.

Si lo prefiere, puede trabajar en modo flujo, que es como una página HTML tradicional, es decir, los elementos se sitúan de arriba a abajo. Cada vista tiene sus ventajas: con el modo cuadrícula es más fácil colocar los elementos. En modo flujo, es más fácil agregar texto estático. Para obtener más información, vea [Colocar elementos HTML en la vista Diseño](#).

En este tutorial, trabajará en modo cuadrícula. Aprenderá a colocar texto estático en este modo.

## Agregar controles a una página de formularios Web Forms

Los controles de formularios Web Forms se denominan "controles de servidor" porque, cuando se ejecuta la página, se crea una instancia de los controles en el código del servidor como parte de la clase de la página. Cuando los usuarios interactúan con los controles, por ejemplo, cuando el usuario hace clic en un control de botón de formularios Web Forms, el código asociado al control se ejecuta en el servidor después de enviar la página. En el código del servidor, puede escribir controladores de eventos para los controles de servidor, establecer sus propiedades, etc.

No todos los elementos de una página de formularios Web Forms son controles de servidor. Por ejemplo, de manera predeterminada, el texto HTML estático no es un control de servidor y no puede controlarlo en el código del servidor. Los controles HTML estándar (por ejemplo, un botón de envío de HTML) no son controles de servidor de manera predeterminada, es decir, no son visibles como controles de la clase en el código del servidor. (Los elementos HTML se pueden programar en secuencias de comandos de cliente, como en cualquier página HTML).

Por lo tanto, para trabajar con controles en una página de formularios Web Forms, es necesario agregarlos como controles de servidor. Existen controles de servidor de dos tipos:

- **Controles de servidor HTML** Son elementos HTML marcados (que se convierten) para que se puedan programar en código de servidor. Normalmente, los elementos HTML sólo se convierten en controles de servidor HTML si existe alguna razón que justifique su programación desde el código del servidor.
- **Controles de servidor Web** Son controles específicos de los formularios Web Forms que proporcionan más funciones que los controles de servidor HTML y que no se corresponden directamente con los elementos HTML.

Para obtener más información sobre estos controles, vea [Introducción a los controles de servidor ASP.NET](#).

En esta sección, agregará un control de cada tipo.

### Para agregar un control de servidor HTML a una página de formularios Web Forms

1. Haga clic en la ficha **Diseño** en la parte inferior de la ventana para cambiar a la vista **Diseño**.
2. En la ficha **HTML** del **Cuadro de herramientas**, arrastre un elemento **Campo de texto** a la página.

Cambie a la vista HTML para ver que la acción ha agregado una etiqueta similar a la siguiente:

```
<INPUT type="text">
```

3. Cambie a la vista **Diseño**.

4. Convierta el elemento de texto HTML en un control de servidor haciendo clic con el botón secundario del *mouse* y eligiendo **Ejecutar como control del servidor**.

Aparecerá un glifo (🔗) en la esquina superior izquierda del control para indicar que se trata de un control de servidor. La operación de conversión de un elemento HTML lo transforma en un control de servidor **HtmlInputText**.

Como parte de su conversión, se agregan dos atributos al elemento HTML:

- El atributo **id** identifica el control en el código. Para el control de servidor **HtmlInputText**, este atributo está establecido de forma predeterminada en el nombre **Text1**.
- El atributo **runat** está establecido en **server**. Esto marca el elemento como control de servidor y lo hace visible para el código de servidor del formulario Web Forms como un elemento programable.

#### Para agregar un control de servidor Web a una página de formularios Web Forms

- En la ficha **Web Forms** del **Cuadro de herramientas** (no en la ficha **HTML**), arrastre un control **Button** de servidor Web a la página.

**Sugerencia** La ficha **Web Forms** sólo está disponible cuando el diseñador se encuentra en la vista **Diseño**.

Este paso crea un elemento de control de servidor Web en el diseñador más que un botón HTML. Si cambia a la vista HTML, verá lo siguiente:

```
<asp:Button id="Button1" runat="server"></asp:Button>
```

Este elemento no corresponde directamente a un elemento HTML. En su lugar, cuando se ejecuta la página, se crea y se procesa una instancia del control **Button** de servidor Web. Durante el procesamiento de la página, el control va mostrando algunos elementos HTML en la página (en este caso, un elemento HTML `<input type=submit>`).

#### Agregar texto HTML a la página

En el modo cuadrícula, es muy sencillo colocar los controles. ¿Pero qué ocurre con el texto HTML estático? Como todos los elementos de la página se sitúan con las coordenadas x e y, no puede escribir el texto en la página simplemente donde desea que aparezca, como lo haría en el modo de flujo.

La solución es agregar un control **Label** de HTML en el que puede incluir el texto estático. Después ya puede colocar la etiqueta (que es un elemento `<DIV>`) en cualquier lugar de la página.

#### Para agregar texto estático en modo cuadrícula

1. En la ficha **HTML** del Cuadro de herramientas, arrastre un control **Label** a la página. Colóquelo y asígnele el tamaño que se ajuste al texto que desea escribir.
2. Haga clic en la etiqueta para seleccionarla y, a continuación, haga clic en ella de nuevo. (Haga esto despacio para no hacer un doble clic en el elemento).

La etiqueta aparece en modo de edición de texto, por lo que aparece marcada con un borde sombreado.

3. Escriba el texto estático que desee. Por ejemplo, escriba "Página de formularios Web Forms de ejemplo".
4. Seleccione el texto y utilice las herramientas de la barra de herramientas Formato para establecer el formato del bloque de texto, la fuente, el tamaño, etc.
5. Haga clic fuera de la etiqueta para salir del modo de edición de texto.

#### Crear un controlador de eventos

Los controles de servidor de las páginas de formularios Web Forms pueden producir diversos eventos, muchos de los cuáles se inician por acciones que realiza el usuario en el explorador. Por ejemplo, un control **Button** de servidor Web puede provocar un evento **Click** cuando un usuario haga clic en un botón de la página.

El código para gestionar el evento provocado se ejecuta en el servidor. Cuando el usuario hace clic en un botón, la página se envía al servidor. El marco de trabajo de páginas ASP.NET analiza la información del evento y si existe un controlador de eventos que se corresponda con el evento, se llama al código inmediatamente. Cuando el código finalice, la página se vuelve a enviar al explorador con cualquier cambio producido por el código del controlador de eventos.

#### Para crear un controlador de eventos para el control Button de servidor Web

1. Haga doble clic en el control **Button** de servidor Web.

El diseñador abrirá el archivo de clase del formulario actual y creará un esquema del controlador de eventos para el evento **Click** del control Button. El código será como el que aparece a continuación:

```
// Visual J#
private void Button1_Click(Object sender, System.EventArgs e)
{

}
```

2. Escriba código en el método para mostrar un mensaje en el control de servidor **HtmlInputText** estableciendo su propiedad **Value**. Por ejemplo:

```
// Visual J#
private void Button1_Click(Object sender, System.EventArgs e)
{
    Text1.set_Value ( "Hello, Web Forms!");
}
```

## Generar y ejecutar la página de formularios Web Forms

Antes de ejecutar la página de formularios Web Forms, debe compilarse el archivo de clase de la página. A continuación, se podrá ver la página en cualquier explorador.

### Para generar y ejecutar la página

1. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* (ratón) en la página WebForm1.aspx y seleccione **Ver en el explorador**.

**Sugerencia** Si se encuentra en la vista Diseño, presione CTRL+F8 para ejecutar el comando **Ver en el explorador**.

Visual Studio compila la página y la muestra en la ficha **Examinar**.

2. Haga clic en el botón de la página de formularios Web Forms.

Aparecerá el texto "Hello, Web Forms!" en el cuadro de texto.

3. Para detener la ejecución del formulario y volver al modo de diseño, cierre la ficha **Examinar**.

## Pasos siguientes

Ésta es una página de formularios Web Forms sencilla, pero ilustra los pasos fundamentales necesarios para crear y editar formularios Web Forms. Desde este punto, puede comenzar a explorar las herramientas incluidas en Visual Studio para ayudarle a crear aplicaciones Web sofisticadas. Sugerencias para una exploración más a fondo incluyen las siguientes:

- Investigar los tipos de controles que se pueden usar en las páginas de formularios Web Forms. Para obtener más información, vea [Introducción a los controles de servidor ASP.NET](#).
- Haga pruebas en el modo flujo en lugar de en el modo cuadrícula. Para obtener más información, vea [Colocar elementos HTML en la vista Diseño](#).
- Comprobar la entrada de los usuarios en la página de formularios Web Forms mediante controles de validación. Para obtener más información, vea [Introducción a la validación de datos proporcionados por el usuario en formularios Web Forms](#).
- Explorar el uso de controles más complejos, tales como el control **Calendar** de servidor Web. Para obtener más información, vea [Calendar \(Control de servidor Web\)](#).
- Enlazar el formulario con un origen de datos y mostrar información en controles individuales o en controles complejos del formulario. Para obtener más información, vea [Acceso a datos en páginas de formularios Web Forms](#). Controles como **DataList** y **DataGrid** se proporcionan para facilitar la implementación de interfaces de usuario controladas por datos. Para obtener más información, vea [Introducción al control DataList de servidor Web](#) e [Introducción al control DataGrid de servidor Web](#).

## **Vea también**

[Introducción a las páginas de formularios Web Forms](#) | [Crear y administrar páginas de formularios Web Forms](#) | [Controles de servidor ASP.NET](#) | [Button \(Controles de servidor Web\)](#) | [Tutoriales sobre los formularios Web Forms](#)

# Tutorial: crear una aplicación Web utilizando Visual J#

En este tutorial se escribirá una aplicación con formularios Web Forms en Visual J# y se incorporará un componente de objeto comercial que ofrezca un descuento del 10% por las compras. Se creará el objeto comercial como clase de Visual J# que implemente las reglas comerciales mediante el cálculo del precio con descuento de un libro específico.

## Crear el proyecto de aplicación Web

En esta sección, se crea un proyecto de aplicación Web de Visual J# que representa la interfaz de usuario de la aplicación. En la página de formularios Web Forms, se colocarán componentes **DataSet**, **DataGrid** y **DataView** para ver una tabla específica de la base de datos.

Se agregará también un componente en el que se creará el objeto de acceso a datos que contiene el origen de datos.

### Para crear el formulario Web Forms

1. En el menú **Archivo**, haga clic en **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. En el panel Tipo de proyecto, haga clic en **Proyectos de Visual J#** y, en el panel Plantillas, elija **Aplicación Web ASP.NET**.
3. Denomine al proyecto **MyWebForm** y acepte el servidor Web predeterminado o escriba el nombre de un servidor en el cuadro de edición Ubicación.
4. Haga clic en **Aceptar**.

El asistente para aplicaciones creará los archivos de proyecto necesarios, incluidos los siguientes:

- WebForm1.aspx: contiene la representación visual del formulario Web Forms.
- WebForm1.aspx.jsl: archivo de código subyacente que contiene el código para el control de eventos y otras tareas del programa. Para ver este archivo en el Explorador de soluciones, haga clic en el icono **Mostrar todos los archivos** y, a continuación, expanda el nodo WebForm1.aspx. Si no está abierto el Explorador de soluciones, elija **Explorador de soluciones** en el menú **Ver**.

Para obtener información detallada sobre la creación de un nuevo proyecto de aplicación Web, vea [Crear proyectos Web Forms](#).

**Nota** Si utiliza una base de datos de SQL Server en otro equipo o un equipo con Terminal Server, siga las instrucciones que encontrará en [Utilizar servidores SQL Server en otros equipos](#).

### Para agregar el componente

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nombre del proyecto.
2. En el menú contextual, haga clic en **Agregar** y, después, en **Agregar componente**.

Aparece el cuadro de diálogo **Agregar nuevo elemento** y se selecciona de forma predeterminada en el panel derecho la **Clase de componentes**.

3. Acepte el nombre predeterminado (**Component1**) y haga clic en **Abrir**.

A menos que elija otro nombre para el componente, esta operación crea un archivo nuevo en el proyecto denominado Component1.jsl. El Diseñador de componentes abre Component1.jsl en la vista Diseño.

## Crear el componente de acceso a datos

Se puede utilizar un conjunto de datos para enlazar los valores de datos con el HTML que se transmite al cliente. Se puede crear en el propio formulario Web Forms o en el componente. Para simular las situaciones reales, se creará en el componente. También se escribirá el código necesario para rellenar el conjunto de datos con información procedente de la base de datos.

### Para agregar y configurar un origen de datos

1. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
2. En la ficha **Datos** del Cuadro de herramientas, arrastre **sqlDataAdapter** a la superficie del Diseñador de componentes.

Aparece el Asistente para la configuración del adaptador de datos.

3. Haga clic en **Siguiente** para pasar a la página **Elegir la conexión de datos** y, a continuación, haga clic en **Nueva conexión**.

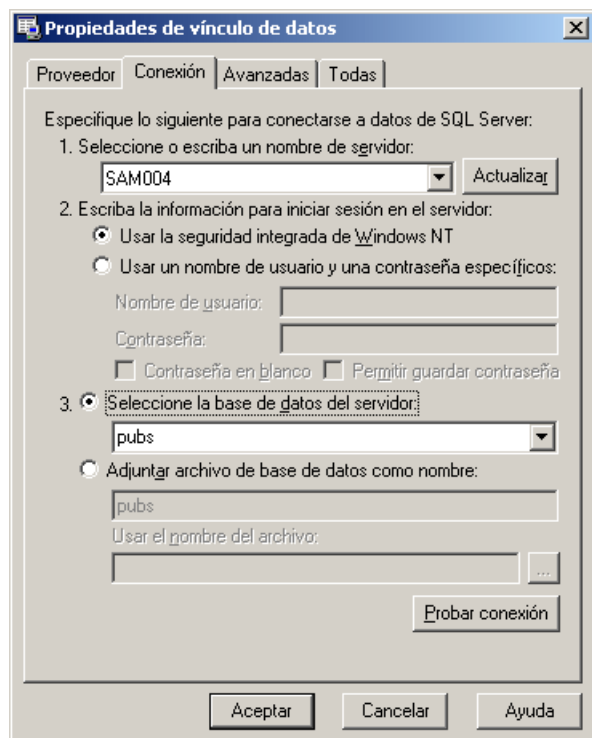
Aparecerá el cuadro de diálogo **Propiedades de vínculo de datos**.

4. En la ficha **Conexión** del cuadro de diálogo **Propiedades de vínculo de datos**:
  - a. Escriba el nombre del servidor donde está almacenada la base de datos pubs.
  - b. Escriba la información de inicio de sesión del servidor.
  - c. Seleccione **pubs** en la lista de bases de datos.
  - d. Pruebe el vínculo de datos haciendo clic en el botón **Probar conexión**.
  - e. Haga clic en **Aceptar** para volver al asistente.

El nombre de la conexión de datos aparece en la lista desplegable.

**Nota** Si no conoce el nombre y la contraseña de usuario para la base de datos pubs de SQL Server, póngase en contacto con el administrador de la base de datos.

### Propiedades de vínculo de datos

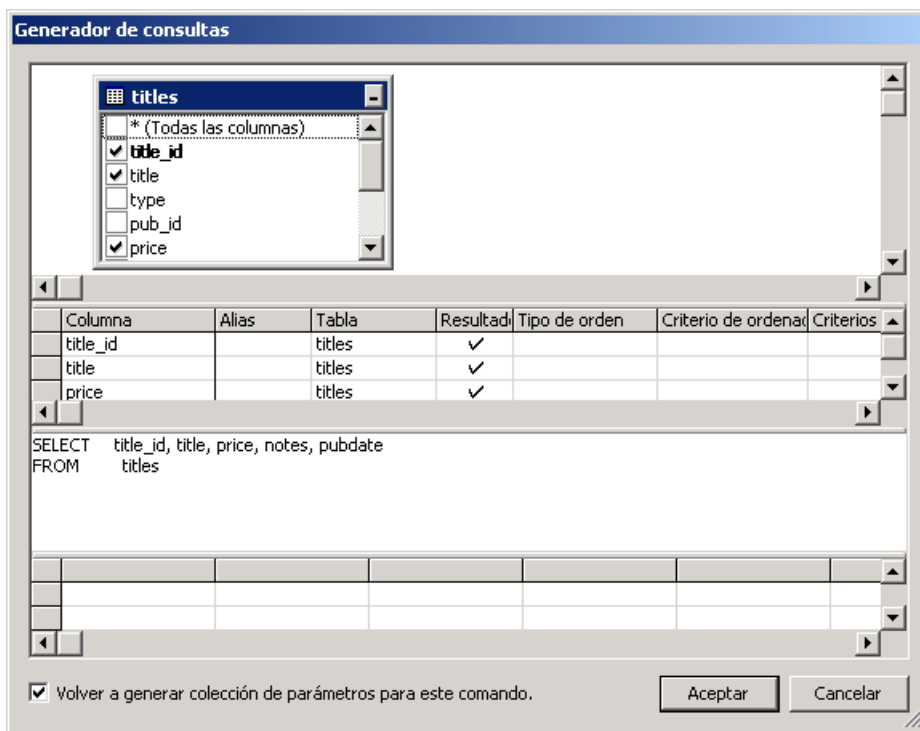


5. Haga clic en **Siguiente** para pasar a la página **Elija un tipo de consulta**.
6. Seleccione **Usar instrucciones SQL** y, a continuación, haga clic en **Siguiente**.
7. En la página **Generar las instrucciones SQL**, haga clic en **Generador de consultas**.
8. En la ficha **Tablas** del cuadro de diálogo **Agregar tabla**, haga clic en **titles**, luego en **Agregar** y, finalmente, en **Cerrar**.

Esta acción agregará la tabla **titles** a la consulta SQL.

9. En el cuadro de diálogo **Generador de consultas**, que se muestra en la siguiente ilustración, seleccione las casillas de verificación **title\_id**, **title**, **price**, **notes** y **pubdate** y, a continuación, haga clic en **Aceptar** para generar la instrucción SQL y volver al Asistente para la configuración del adaptador de datos.

### Generador de consultas



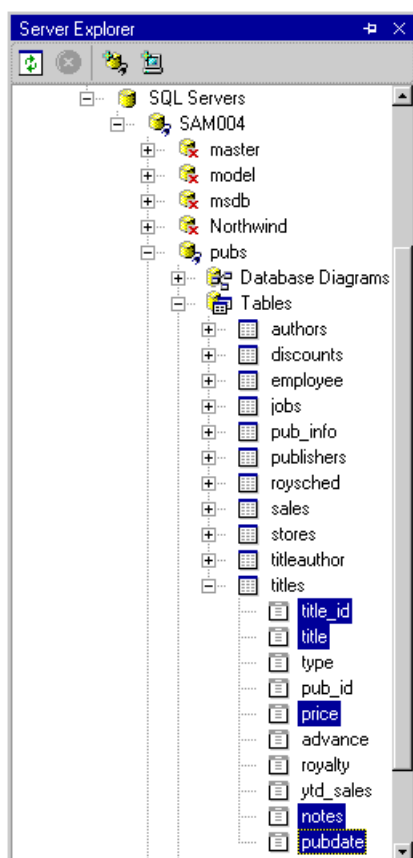
En la página **Generar las instrucciones SQL** se muestra la instrucción SQL que se ha generado.

- Haga clic en **Finalizar** para salir del Asistente para la configuración del adaptador de datos y conectar el origen de datos con la tabla **titles**. Se agregan los siguientes objetos en la superficie de diseño del Diseñador de componentes:

- **sqlDataAdapter1**
- **sqlConnection1**

Otra posibilidad es crear ambos objetos, **sqlDataAdapter1** y **sqlConnection1** mediante el Explorador de servidores. Para ello, expanda el árbol **Servidores SQL** para que muestre el servidor utilizado, la base de datos **pubs** y la tabla **titles**, como se muestra en la ilustración siguiente. Expanda **titles** y seleccione las columnas requeridas, y arrástrelas a la superficie del Diseñador de componentes.

### Explorador de servidores



Puede ver el código generado en el método **InitializeComponent** del archivo Component1.jsl. Para ello, cambie a la vista de código haciendo clic con el botón secundario del *mouse* en Component1.jsl en el Explorador de soluciones, y luego elija **Ver código** en el menú contextual.

11. Guarde el proyecto haciendo clic en **Guardar todo** en el menú **Archivo**.

### Para generar y rellenar el conjunto de datos

1. Si el Diseñador de componentes no está visible, haga doble clic en el archivo Component1.jsl en el Explorador de soluciones.
2. En el menú **Datos**, haga clic en **Generar conjunto de datos**.  
Aparecerá el cuadro de diálogo **Generar conjunto de datos**.
3. Si no está seleccionado el botón de opción **Nuevo**, selecciónelo. En el cuadro de texto correspondiente, escriba un nombre como por ejemplo **myDataSet**.
4. Asegúrese de que la casilla **Agregar este conjunto de datos al diseñador** no está seleccionada y haga clic en **Aceptar**.

El archivo myDataSet.xsd aparece en el Explorador de soluciones.

Para ver el esquema y el código XML que describen **myDataSet**, haga clic en myDataSet.xsd en el Explorador de soluciones. Observe que las fichas **DataSet** y **XML** se encuentran en la esquina inferior izquierda de la superficie del Diseñador XML.

### Enlazar el componente DataGridView

En esta sección se agrega un conjunto de datos a la página de formularios Web Forms, se rellena con datos y se enlaza con el componente **DataGridView**.

#### Para agregar un conjunto de datos al formulario

1. En el Explorador de soluciones, haga doble clic en el archivo WebForm1.aspx para seleccionar la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un objeto **DataSet** desde la ficha **Datos** del Cuadro de herramientas hasta la superficie del Diseñador de Web Forms.

Aparecerá el cuadro de diálogo **Agregar conjunto de datos**.

4. Si no está seleccionado **TypedDataSet**, selecciónelo.
5. Seleccione el nombre del conjunto de datos (**MyWebForm.myDataSet**) en la lista desplegable y haga clic en **Aceptar**.

De esta forma se agrega un conjunto de datos, **myDataSet1**, a la parte no visual de la superficie del Diseñador de Web Forms.

#### Para llenar el conjunto de datos

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en Component1.jsl en el Explorador de soluciones, y seleccione **Ver código** en el menú contextual para cambiar a la vista de código del componente.
2. Agregue el método siguiente a la clase **Component1**, después del constructor **Component1** pero antes del código generado por el Diseñador de componentes:

```
// Visual J#
public class Component1 extends System.ComponentModel.Component
{
    ...
    // Add the following code:
    public void FillDataSet(myDataSet dSet)
    {
        sqlDataAdapter1.Fill(dSet);
        sqlConnection1.Close();
    }
    // End of the new code.
    ...
}
```



3. Cambie a la vista de código del formulario Web Forms; para ello, haga clic con el botón secundario del *mouse* en el archivo WebForm1.aspx en el Explorador de soluciones, y seleccione **Ver código** en el menú contextual.

Se abrirá el archivo de código subyacente, WebForm1.aspx.jsl.

4. Declare un componente en el nivel superior de la clase **WebForm1**, como se indica a continuación:

```
// Visual J#
public class WebForm1 extends System.Web.UI.Page
{
    // Add the following line:
    protected Component1 myComponent = new Component1();
    ...
}
```

5. Modifique el método **Page\_Load** para que llame a `FillDataSet` como sigue:

```
// Visual J#
private void Page_Load (Object sender, System.EventArgs e)
{
    // Add the following code:
    if (! get_IsPostBack())
    {
        myComponent.FillDataSet(myDataSet1);
    }
    // End of the new code.
}
```

Observe que la instrucción que aparece dentro del bloque condicional tiene el valor **true** la primera vez que el explorador llega a la página.

### Para agregar y configurar el componente **DataGrid**

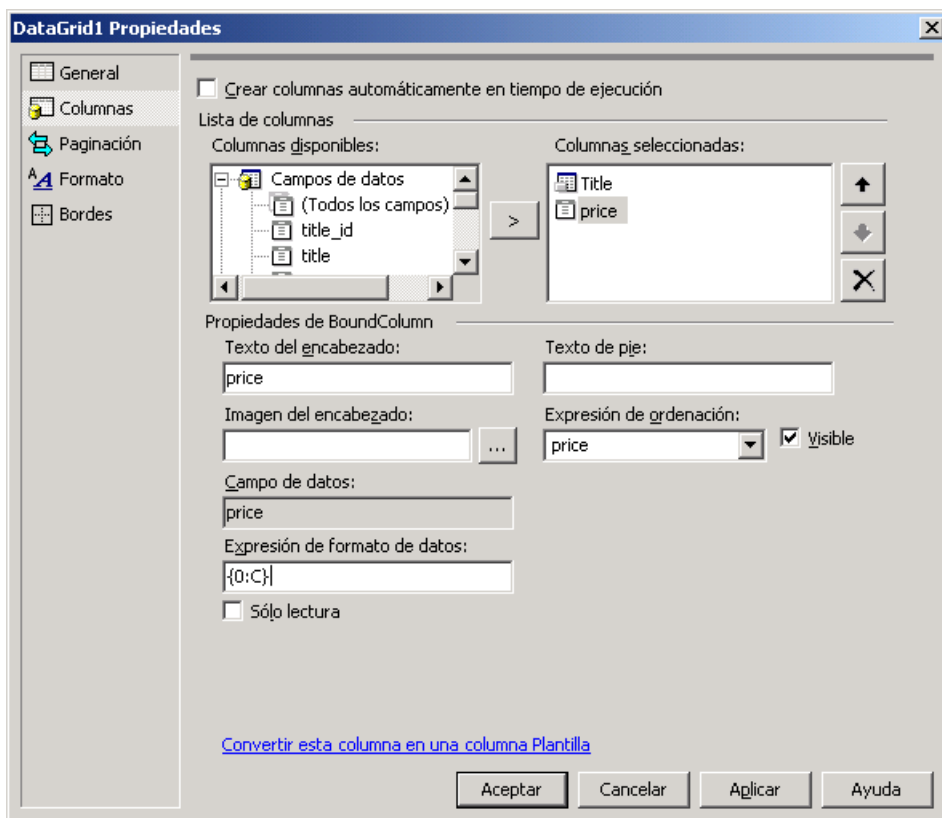
1. En el Explorador de soluciones, haga doble clic en Webform1.aspx para cambiar a la vista Diseño de la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un control **DataGrid** desde la ficha **Web Forms** del Cuadro de herramientas hasta la superficie del Diseñador de Web Forms.
4. Haga clic con el botón secundario del *mouse* en el control **DataGrid** y seleccione **Generador de propiedades** en el menú contextual.

Aparece el cuadro de diálogo **Propiedades de DataGrid1**.

5. En el formulario General, establezca el valor de la propiedad **DataSource** en **myDataSet1**, **DataMember** en **titles** y el **Campo de claves de datos** en **title\_id** (la clave principal de la base de datos).
6. En el formulario **Columnas**:
  - a. Desactive la casilla de verificación **Crear columnas automáticamente en tiempo de ejecución**.
  - b. Desplácese hacia abajo en la lista **Columnas disponibles** hasta el nodo **Columna Botón**.
  - c. Expanda el nodo **Columna Botón** y agregue el botón **Seleccionar** seleccionándolo y haciendo clic en la flecha hacia la derecha que se encuentra entre ambas listas.
  - d. Escriba "Título" en el cuadro **Texto de encabezado**. Establezca la **Expresión de ordenación** en "title" y el **Campo de texto** en "title". Haga clic en **Aplicar**.
  - e. Agregue la columna **price** de la lista **Columnas disponibles** seleccionándola y haciendo clic en la flecha hacia la derecha que se encuentra entre ambas listas.
  - f. En el cuadro **Texto de encabezado**, escriba un encabezado apropiado, como por ejemplo "Precio".
  - g. En el cuadro de texto **Expresión de formato de datos**, escriba la cadena de formato de moneda {0:C}.
  - h. Haga clic en **Aceptar**.

**Nota** Asegúrese de que la casilla de verificación **Visible** está activada para todas las columnas.

### Propiedades de DataGrid1



En el **DataGrid** se muestran únicamente las columnas **Título** y **Precio** del **DataSet**.

7. Cambie a la vista de código de la página de formularios Web Forms. Enlace los datos con las columnas de **DataGrid** agregando una llamada a **DataBind** en el método **Page\_Load**. El método presentará el siguiente aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    if (!get_IsPostBack())
    {
        myComponent.FillDataSet(myDataSet1);
        DataGrid1.DataBind(); // Add this line
    }
}
```

Para obtener más detalles sobre cómo enlazar controles, vea [Acceso a datos en las páginas de formularios Web Forms](#).

### Para probar el proyecto

1. En el menú **Generar**, haga clic en **Generar solución**.
2. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Ahora puede mostrar datos de la base de datos en una página de formulario Web Forms y verla en el explorador. La columna **Títulos** está formada por hipervínculos. Más adelante se podrá hacer clic en uno de los hipervínculos y mostrar los detalles de un libro específico.

### Ver la tabla Titles en el explorador

Title	Price
<a href="#">The Busy Executive's Database Guide</a>	\$19.99
<a href="#">Cooking with Computers: Surreptitious Balance Sheets</a>	\$11.95
<a href="#">You Can Combat Computer Stress!</a>	\$2.99
<a href="#">Straight Talk About Computers</a>	\$19.99
<a href="#">Silicon Valley Gastronomic Treats</a>	\$19.99
<a href="#">The Gourmet Microwave</a>	\$2.99
<a href="#">The Psychology of Computer Cooking</a>	
<a href="#">But Is It User Friendly?</a>	\$22.95
<a href="#">Secrets of Silicon Valley</a>	\$20.00
<a href="#">Net Etiquette</a>	
<a href="#">Computer Phobic AND Non-Phobic Individuals: Behavior Variations</a>	\$21.59
<a href="#">Is Anger the Enemy?</a>	\$10.95
<a href="#">Life Without Fear</a>	\$7.00
<a href="#">Prolonged Data Deprivation: Four Case Studies</a>	\$19.99
<a href="#">Emotional Security: A New Algorithm</a>	\$7.99
<a href="#">Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean</a>	\$20.95
<a href="#">Fifty Years in Buckingham Palace Kitchens</a>	\$11.95
<a href="#">Sushi, Anyone?</a>	\$14.99

## Agregar un panel de detalles al formulario Web Forms

El panel de detalles permite ver información adicional acerca de los libros seleccionados, sin saturar el control **DataGrid** con material ajeno a él.

En esta fase se agregará un control **DataView**, que permite filtrar la tabla para mostrar únicamente la fila seleccionada.

### Para agregar un control DataView


1. Cambie a la vista Diseño de la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un control **DataView** desde la ficha **Datos** hasta la superficie del Diseñador de Web Forms.

De esta forma se agrega un nuevo objeto, **dataView1**, a la parte no visual de la superficie del Diseñador de Web Forms.

4. Seleccione el objeto **dataView1** y muestre sus propiedades haciendo clic en **Ventana Propiedades** en el menú **Ver**.
5. Defina la propiedad **Table** como myDataSet1.titles en la ventana **Propiedades**.

En este paso se agregarán controles **Label** (etiqueta) correspondientes a columnas de datos, y se enlazarán cada **Label** con el control **DataView**.

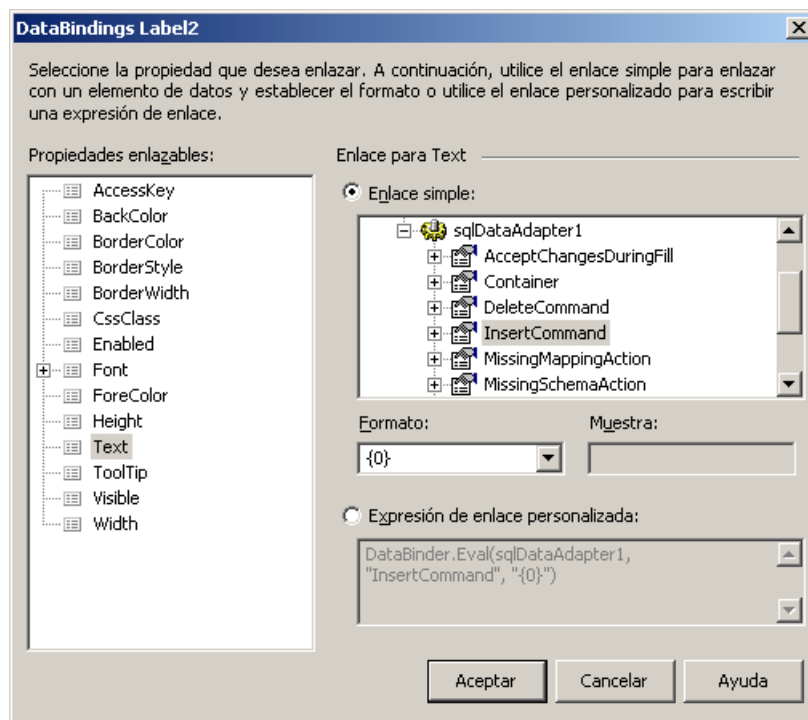
### Para agregar controles Label

1. Para cada campo de detalle utilizado (como por ejemplo **title-id**, **title**, **price** y **pubdate**), arrastre un control **Label** de la ficha **Web Forms** del Cuadro de herramientas a la superficie del Diseñador de Web Forms.
2. Seleccione cada una de las etiquetas y enlázela con un campo, de esta forma:
  - a. Abra la página **Propiedades** de la etiqueta seleccionada haciendo clic en **Ventana Propiedades** en el menú **Ver**.
  - b. Expanda la propiedad **Data**.
  - c. Seleccione la propiedad **DataBindings** haciendo clic en el botón de puntos suspensivos () situado junto a **DataBindings**.

Aparece el cuadro de diálogo **DataBindings** de **Label $n$**  (donde  $n$  es el número de la etiqueta). En la siguiente ilustración se muestra el cuadro de diálogo correspondiente a **Label1**.

- d. Seleccione la propiedad **Text** en **Propiedades enlazables** en el panel izquierdo, y seleccione la columna apropiada del nodo **dataView1** en el panel derecho.

### Propiedades enlazables de una etiqueta



- e. Seleccione el formato adecuado en la lista desplegable **Formato**. Por ejemplo, puede utilizar el formato **Moneda** para **price**, y uno de los formatos de fecha disponibles para **pubdate**.
  - f. Haga clic en **Aceptar**.
3. Agregue texto para describir las etiquetas, precediendo cada una de ellas con una **Etiqueta HTML** que contenga el texto descriptivo (por ejemplo, Id. de título, Título, Fecha pub. y Precio). Para ello, abra el Cuadro de herramientas y arrastre un control **Label** desde la ficha **HTML** a la superficie del Diseñador de Web Forms. Sitúe la etiqueta delante de la etiqueta relacionada correspondiente y modifique su texto de la forma apropiada. Repita este proceso para crear cuatro etiquetas.

En este paso se agregará el código necesario para activar los detalles al hacer clic en el control **DataGrid**.

### Para activar los detalles

1. Haga doble clic en el objeto **DataGrid1**.

De esta forma se agrega un controlador de eventos **DataGrid1\_SelectedIndexChanged** en el archivo de código subyacente y se cambia a la vista de código.

Modifique la definición del controlador de eventos, de la actual:

```
private void DataGrid1_SelectedIndexChanged (Object sender, System.EventArgs e)
```

a:

```
private void DataGrid1_SelectedIndexChanged (Object sender, System.EventArgs e) throws System.Data.StrongTypingException
```

2. En el método **DataGrid1\_SelectedIndexChanged**, agregue el código siguiente para configurar **dataView1.RowFilter** de forma que seleccione únicamente la fila que se desea mostrar:

```
// Visual J#
myComponent.FillDataSet(myDataSet1);
int index = DataGrid1.get_SelectedIndex();
String key = DataGrid1.get_DataKeys().get_Item(index).ToString();
dataView1.set_RowFilter(DataGrid1.get_DataKeyField ()+ "=" + key + "");
```

Conceptualmente, esto es muy parecido a especificar la cláusula WHERE de una consulta SQL y utilizarla para obtener una única fila de la tabla. La propiedad **DataKeys**, especificada al crear **DataGrid**, es un medio para identificar cada fila de forma exclusiva. El valor de la clave de la fila seleccionada se determina asignado el índice del elemento a una clave coincidente. Esta clave, que es un **title\_id** válido, se puede entonces utilizar para seleccionar de forma exclusiva la fila que se desea

utilizar. Esto es así porque **title\_id** es la clave principal de la base de datos. Con otra base de datos o con una clave de datos distinta no es posible garantizar estas restricciones; por tanto, si se desea utilizar esta técnica con tablas que no disponen de una clave principal, se debe modificar el código.

3. Enlace cada una de las etiquetas utilizadas llamando al método **DataBind** con esa etiqueta en el método **DataGrid1\_SelectedIndexChanged**. Sitúe estas llamadas después de efectuar el cambio del filtro de filas:

```
// Visual J#
Label1.DataBind();
Label2.DataBind();
Label3.DataBind();
Label4.DataBind();
```

4. Genere e inicie el proyecto.

Ahora podrá hacer clic en un título y ver los detalles con la información del nuevo título.

## Crear un objeto comercial en Visual J#

En esta sección, se creará el objeto comercial en forma de clase de Visual J#. Esta clase contendrá un método que implementará las reglas comerciales mediante el cálculo del precio con descuento para un libro específico.

### Para crear el objeto comercial en Visual J#

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo del proyecto.
2. En el menú contextual, haga clic en **Agregar** y, después, en **Agregar clase**.

Aparecerá el cuadro de diálogo Agregar nuevo elemento.

3. En el cuadro de diálogo, asigne a la clase el nombre **MyPaymentRules** y haga clic en **Abrir**.
4. Agregue el método siguiente a la clase **MyPaymentRules** en el Editor de código:

```
// Visual J#
public double CalcDiscount(double price)
{
    return 0.9 * price;
}
```

## Utilizar el objeto comercial

En esta sección se agregará una nueva etiqueta enlazada a la página de formularios Web Forms. En la nueva etiqueta se podrá leer "Your Price", que incluye el 10% de descuento.

### Para implementar el objeto comercial

1. Arrastre un control **Label** del Cuadro de herramientas a la superficie del Diseñador de Web Forms (se le asignará el ID **Label5**).

En esta etiqueta se mostrará el precio con descuento.

2. Enlace la propiedad text de **Label5** con la columna price de **DataView1**

No es necesario utilizar un formato específico porque dicho formato se especificará en el código.

3. Arrastre un control **HTML Label** del Cuadro de herramientas a la superficie del Diseñador de Web Forms y sitúelo delante de **Label5**.
4. Cambie el texto de la etiqueta por "Your price".
5. Vea el archivo WebForm1.aspx.jsl y agregue el código siguiente al final del método **DataGrid1\_SelectedIndexChanged**:

```
// Visual J#
// Declare an instance of the business object:
MyPaymentRules pr = new MyPaymentRules();
// Invoke the CalcDiscount Method:
try
{
```

```

        System.Decimal price = myDataSet1.get_titles().get_Item(index).get_price();
        Label5.set_Text(String.Format("{0:C}", (System.Double)pr.CalcDiscount(System.Convert.To
Double(price))));
    }
    catch(Exception ex)
    {
        // If the price is blank, display a message:
        Label4.set_Text("Price is not available for this item.");
        Label5.set_Text("Discount is not available for this item.");
    }
}

```

6. Guarde el proyecto y genere y ejecute la aplicación.

Al hacer clic en cualquier título de la tabla, se obtienen los detalles del libro, incluido el precio con descuento (Your Price) tal como se muestra en la ilustración siguiente.

### Información detallada acerca de un libro

Title ID:	TC7777
Title:	Sushi, Anyone?
Pub. date:	6/12/1991
Price:	\$14.99
Your Price	\$13.49

## Implementación

En este paso se implementará el proyecto en un servidor de producción específico. Para ello, copie el proyecto en el servidor de producción.

### Para copiar el proyecto en un servidor de producción

1. En el menú **Proyecto**, haga clic en **Copiar proyecto**.

Aparece el cuadro de diálogo **Copiar proyecto**.

2. En el cuadro de texto **Carpeta de proyecto de destino**, escriba el nombre del servidor y la carpeta de proyecto; por ejemplo:

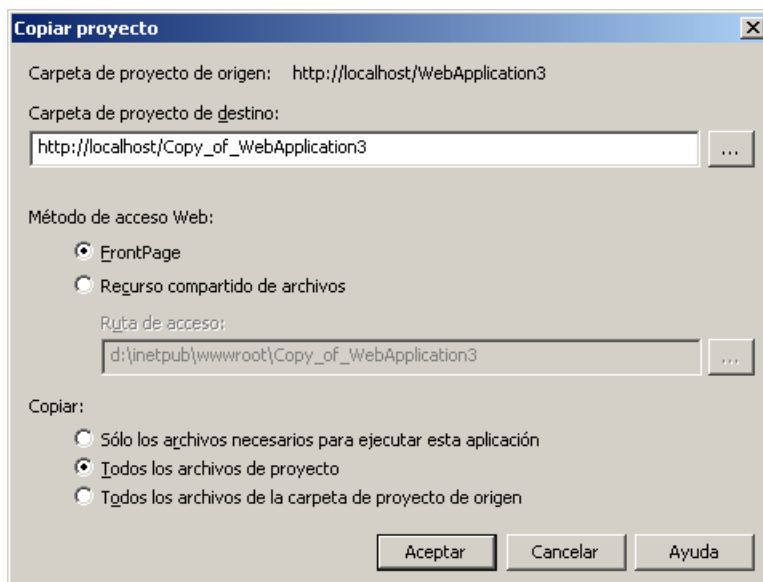
**http://ProductionServerName/MyWebForm/**

3. En la sección **Copiar**, seleccione uno de los siguientes botones de opción:

- **Sólo los archivos necesarios para ejecutar la aplicación.**
- **Todos los archivos del proyecto.**
- **Todos los archivos de la carpeta de proyecto de origen.**

4. Haga clic en **Aceptar** para iniciar el proceso de copia.

**Cuadro de diálogo Copiar proyecto.**



## Vea también

[Extensiones administradas de C++](#) | [Servicios Web XML en Visual Studio](#) | [Tutorial: crear una aplicación distribuida con Visual J#](#) | [DataGrid \(Control Web\)](#) | [Tutoriales sobre los formularios Web Forms](#) | [Seguridad de una base de datos](#)

# Tutorial: crear una aplicación Web utilizando un objeto comercial de otro fabricante con Visual J#

En este tutorial se escribirá una aplicación con formularios Web Forms en Visual J# y se incorporará un componente de objeto comercial que ofrezca un descuento del 10% por las compras.

Para explorar la interoperabilidad entre los diversos lenguajes, se creará el objeto comercial como proyecto DLL independiente mediante las Extensiones administradas de C++. A continuación, se utilizará la DLL en el proyecto de aplicación Web. Este enfoque simula la situación habitual de utilizar un componente ejecutable escrito por un tercero.

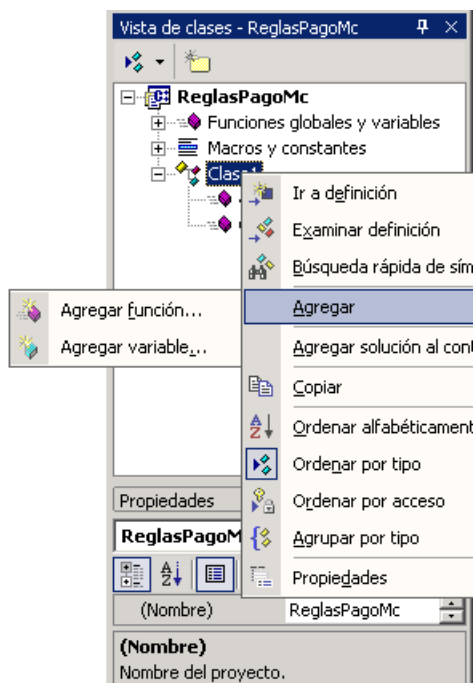
## Crear un objeto comercial externo con Extensiones administradas de C++

En esta sección, se creará un objeto comercial que ofrece un descuento del 10% en compras de libros. Más adelante se creará una aplicación con formularios Web Forms que utilizará dicho objeto.

### Para crear el objeto comercial

1. En el menú **Archivo**, haga clic en **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. En el panel Tipos de proyecto, haga clic en la carpeta **Proyectos de Visual C++** y, en el panel Plantillas, haga clic en el icono **Biblioteca de clases (.NET)**.
3. Asigne al proyecto el nombre McPaymentRules . Después de hacer clic en **Aceptar**, se crearán los archivos de código fuente y de encabezados.
4. En Vista de clases, expanda el nodo de proyecto McPaymentRules y haga clic con el botón secundario en el nodo Class1.
5. En el menú contextual, haga clic en **Agregar** y, después, en **Agregar función**.

### Agregar función



Aparecerá el Asistente para agregar funciones miembro.

6. En dicho asistente, rellene los cuadros con la información siguiente:
  - **Nombre de la función:** CalcDiscount
  - **Tipo de valor devuelto:** double
  - **Tipo de parámetro:** double
  - **Nombre de parámetro:** price.
7. Haga clic en **Agregar** para agregar la función miembro.
8. Haga clic en **Finalizar** para crear la función.



El código de la función aparecerá en el editor de código como parte del archivo McPaymentRules.h.

### Asistente para agregar funciones miembro

9. Si no ve el código de la función, expanda la clase **McPaymentRules** en la Vista de clases hasta que se vea el nodo de función **CalcDiscount(double price)**. Haga clic con el botón secundario en el nodo de la función y seleccione **Ir a definición**. Aparece el siguiente código en el editor:

```
double CalcDiscount(double price)
{
    return 0;
}
```

10. Reemplace la instrucción return por:

```
return price * 0.9;
```

11. Guarde el proyecto haciendo clic en **Guardar todo** en el menú **Archivo**.  
12. En el menú **Generar**, haga clic en **Generar solución** para crear el archivo McPaymentRules.dll.

Más adelante se utilizará este componente en el tutorial.

### Crear el proyecto de aplicación Web

En esta sección, se crea un proyecto de aplicación Web de Visual J# que representa la interfaz de usuario de la aplicación. En la página de formularios Web Forms, se colocarán componentes **DataSet**, **DataGrid** y **DataView** para ver una tabla específica de la base de datos.

Se agregará también un componente en el que se creará el objeto de acceso a datos que contiene el origen de datos.

#### Para crear el formulario Web Forms

1. En el menú **Archivo**, haga clic en **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. En el panel Tipo de proyecto, haga clic en **Proyectos de Visual J#** y, en el panel Plantillas, elija **Aplicación Web ASP.NET**.
3. Denomine al proyecto **MyWebForm** y acepte el servidor Web predeterminado o escriba el nombre de un servidor en el cuadro de edición Ubicación.
4. Haga clic en **Aceptar**.

El asistente para aplicaciones creará los archivos de proyecto necesarios, incluidos los siguientes:

- WebForm1.aspx: contiene la representación visual del formulario Web Forms.

- WebForm1.aspx.jsl: archivo de código subyacente que contiene el código para el control de eventos y otras tareas del programa. Para ver este archivo en el Explorador de soluciones, haga clic en el icono **Mostrar todos los archivos** y, a continuación, expanda el nodo WebForm1.aspx. Si no está abierto el Explorador de soluciones, elija **Explorador de soluciones** en el menú **Ver**.

Para obtener información detallada sobre la creación de un nuevo proyecto de aplicación Web, vea [Crear proyectos Web Forms](#).

**Nota** Si utiliza una base de datos de SQL Server en otro equipo o en un equipo con Terminal Server, siga las instrucciones que encontrará en [Utilizar servidores SQL Server en otros equipos](#).

### Para agregar el componente

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nombre del proyecto.
2. En el menú contextual, haga clic en **Agregar** y, después, en **Agregar componente**.

Aparece el cuadro de diálogo **Agregar nuevo elemento** y se selecciona de forma predeterminada en el panel derecho la **Clase de componentes**.

3. Acepte el nombre predeterminado (**Component1**) y haga clic en **Abrir**.

A menos que elija otro nombre para el componente, esta operación crea un archivo nuevo en el proyecto denominado Component1.jsl. El Diseñador de componentes abre Component1.jsl en la vista Diseño.

### Crear el componente de acceso a datos

Se puede utilizar un conjunto de datos para enlazar los valores de datos con el HTML que se transmite al cliente. Se puede crear en el propio formulario Web Forms o en el componente. Para simular las situaciones reales, se creará en el componente. También se escribirá el código necesario para rellenar el conjunto de datos con información procedente de la base de datos.

### Para agregar y configurar un origen de datos

1. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
2. En la ficha **Datos** del Cuadro de herramientas, arrastre **sqlDataAdapter** a la superficie del Diseñador de componentes.

Con ello se iniciará el Asistente para la configuración del adaptador de datos.

3. Haga clic en **Siguiente** para pasar a la página **Elegir la conexión de datos** y, a continuación, haga clic en **Nueva conexión**.

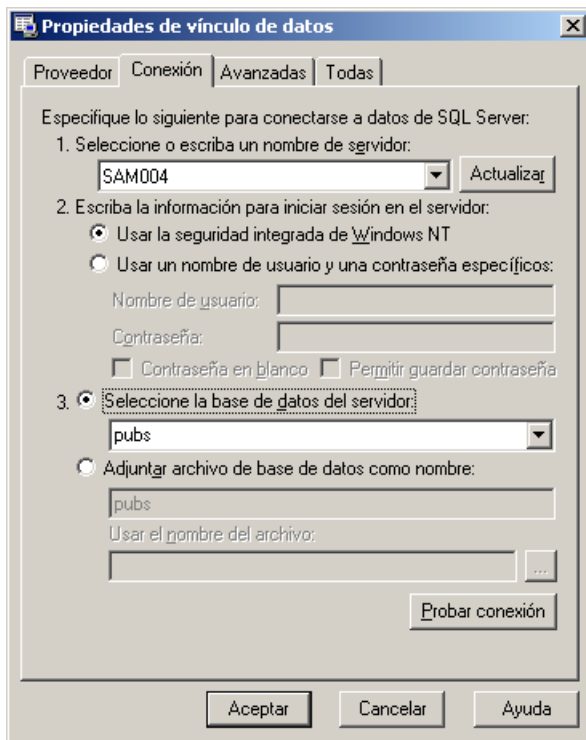
Se abre el cuadro de diálogo **Propiedades de vínculo de datos**.

4. En la ficha **Conexión** del cuadro de diálogo **Propiedades de vínculo de datos**:
  - a. Escriba el nombre del servidor donde está almacenada la base de datos pubs.
  - b. Escriba la información de inicio de sesión del servidor.
  - c. Seleccione **pubs** en la lista de bases de datos.
  - d. Pruebe el vínculo de datos haciendo clic en el botón **Probar conexión**.
  - e. Haga clic en **Aceptar** para volver al asistente.

El nombre de la conexión de datos aparece en la lista desplegable.

**Nota** Si no conoce el nombre y la contraseña de usuario para la base de datos pubs de SQL Server, póngase en contacto con el administrador de la base de datos.

### Propiedades de vínculo de datos

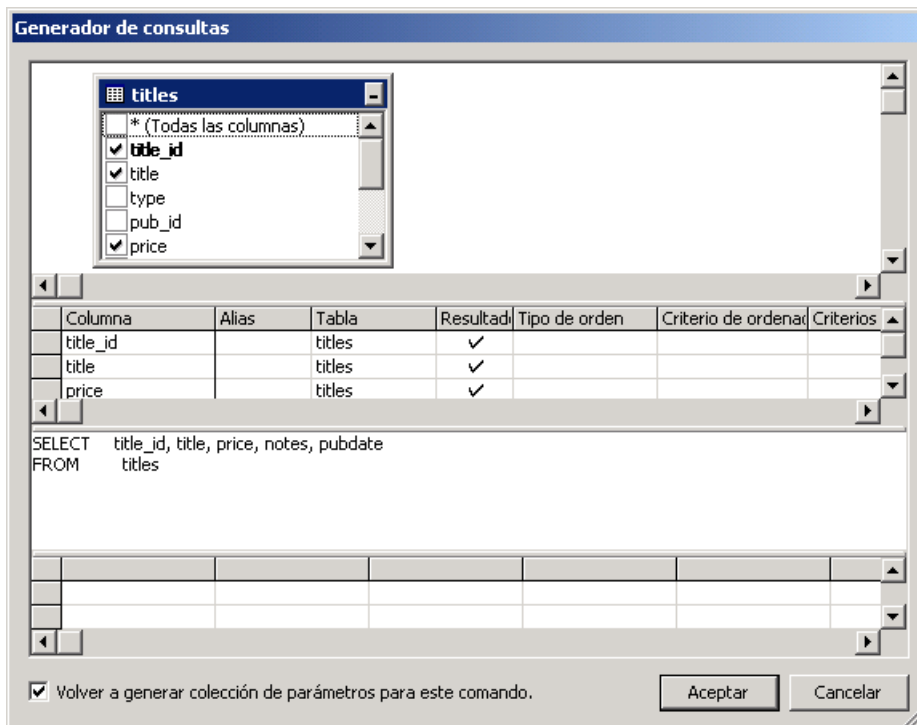


5. Haga clic en **Siguiente** para pasar a la página **Elija un tipo de consulta**.
6. Seleccione **Usar instrucciones SQL** y, a continuación, haga clic en **Siguiente**.
7. En la página **Generar las instrucciones SQL**, haga clic en **Generador de consultas**.
8. En la ficha **Tablas** del cuadro de diálogo **Agregar tabla**, haga clic en **titles**, luego en **Agregar** y, finalmente, en **Cerrar**.

Esta acción agregará la tabla **titles** a la consulta SQL.

9. En el cuadro de diálogo **Generador de consultas**, que se muestra en la siguiente ilustración, seleccione las casillas de verificación **title\_id**, **title**, **price**, **notes** y **pubdate** y, a continuación, haga clic en **Aceptar** para generar la instrucción SQL y volver al Asistente para la configuración del adaptador de datos.

### Generador de consultas



En la página **Generar las instrucciones SQL** se muestra la instrucción SQL que se ha generado.

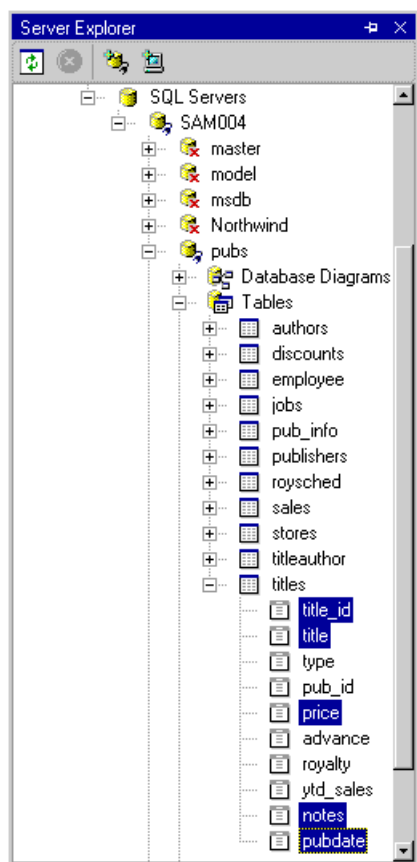
10. Haga clic en **Finalizar** para salir del Asistente para la configuración del adaptador de datos y conectar el origen de datos con la tabla **titles**. Se agregan los siguientes objetos en la superficie de diseño del Diseñador de componentes:

- **sqlDataAdapter1**

- **sqlConnection1**

Otra posibilidad es crear ambos objetos, **sqlDataAdapter1** y **sqlConnection1** mediante el Explorador de servidores. Para ello, expanda el árbol **Servidores SQL** para que muestre el servidor utilizado, la base de datos **pubs** y la tabla **titles**, como se muestra en la ilustración siguiente. Expanda **titles** y seleccione las columnas requeridas, y arrástrelas a la superficie del Diseñador de componentes.

### Explorador de servidores



Puede ver el código generado en el método **InitializeComponent** del archivo Component1.jsl. Para ello, cambie a la vista de código haciendo clic con el botón secundario del *mouse* en Component1.jsl en el Explorador de soluciones, y luego elija **Ver código** en el menú contextual.

11. Guarde el proyecto haciendo clic en **Guardar todo** en el menú **Archivo**.

### Para generar y rellenar el conjunto de datos

1. Si el Diseñador de componentes no está visible, haga doble clic en el archivo Component1.jsl en el Explorador de soluciones.
2. En el menú **Datos**, haga clic en **Generar conjunto de datos**.  
Aparecerá el cuadro de diálogo **Generar conjunto de datos**.
3. Si no está seleccionado el botón de opción **Nuevo**, selecciónelo. En el cuadro de texto correspondiente, escriba un nombre como por ejemplo **myDataSet**.
4. Asegúrese de que la casilla **Agregar este conjunto de datos al diseñador** no está seleccionada y haga clic en **Aceptar**.

El archivo myDataSet.xsd aparece en el Explorador de soluciones.

Para ver el esquema y el código XML que describen myDataSet, haga clic en myDataSet.xsd en el Explorador de soluciones. Observe que las fichas **DataSet** y **XML** se encuentran en la esquina inferior izquierda de la superficie del Diseñador XML.

### Enlazar el componente DataGridView

En esta sección se agrega un conjunto de datos a la página de formularios Web Forms, se rellena con datos y se enlaza con el componente **DataGridView**.

### Para agregar un conjunto de datos al formulario

1. En el Explorador de soluciones, haga doble clic en el archivo WebForm1.aspx para seleccionar la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un objeto **DataSet** desde la ficha **Datos** del Cuadro de herramientas hasta la superficie del Diseñador de Web Forms.

Aparecerá el cuadro de diálogo **Agregar conjunto de datos**.

4. Si no está seleccionado **TypedDataSet**, selecciónelo.
5. Seleccione el nombre del conjunto de datos (**MyWebForm.myDataSet**) en la lista desplegable y haga clic en **Aceptar**.

De esta forma se agrega un conjunto de datos, **myDataSet1**, a la parte no visual de la superficie del Diseñador de Web Forms.

### Para llenar el conjunto de datos

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en Component1.jsl en el Explorador de soluciones, y seleccione **Ver código** en el menú contextual para cambiar a la vista de código del componente.
2. Agregue el método siguiente a la clase **Component1**:

```
// Visual J#
public class Component1 extends System.ComponentModel.Component
{
    ...
    // Add the following code:
    public void FillDataSet(myDataSet dSet)
    {
        sqlDataAdapter1.Fill(dSet);
    sqlConnection1.Close();
    }
    // End of the new code.
    ...
}
```

3. Cambie a la vista de código del formulario Web Forms; para ello, haga clic con el botón secundario del *mouse* en el archivo WebForm1.aspx en el Explorador de soluciones, y seleccione **Ver código** en el menú contextual.

Se abrirá el archivo de código subyacente, WebForm1.aspx.jsl.

4. Declare un componente en el nivel superior de la clase **WebForm1**, como se indica a continuación:

```
// Visual J#
public class WebForm1 extends System.Web.UI.Page
{
    // Add the following line:
    protected Component1 myComponent = new Component1();
    ...
}
```

5. Modifique el método **Page\_Load** para que llame a `FillDataSet` como sigue:

```
// Visual J#
private void Page_Load (Object sender, System.EventArgs e)
{
    // Add the following code:
    if (!get_IsPostBack())
    {
        myComponent.FillDataSet(myDataSet1);
    }
    // End of the new code.
}
```

Observe que la instrucción que aparece dentro del bloque condicional tiene el valor **true** la primera vez que el explorador llega a la página.

### Para agregar y configurar el componente DataGrid

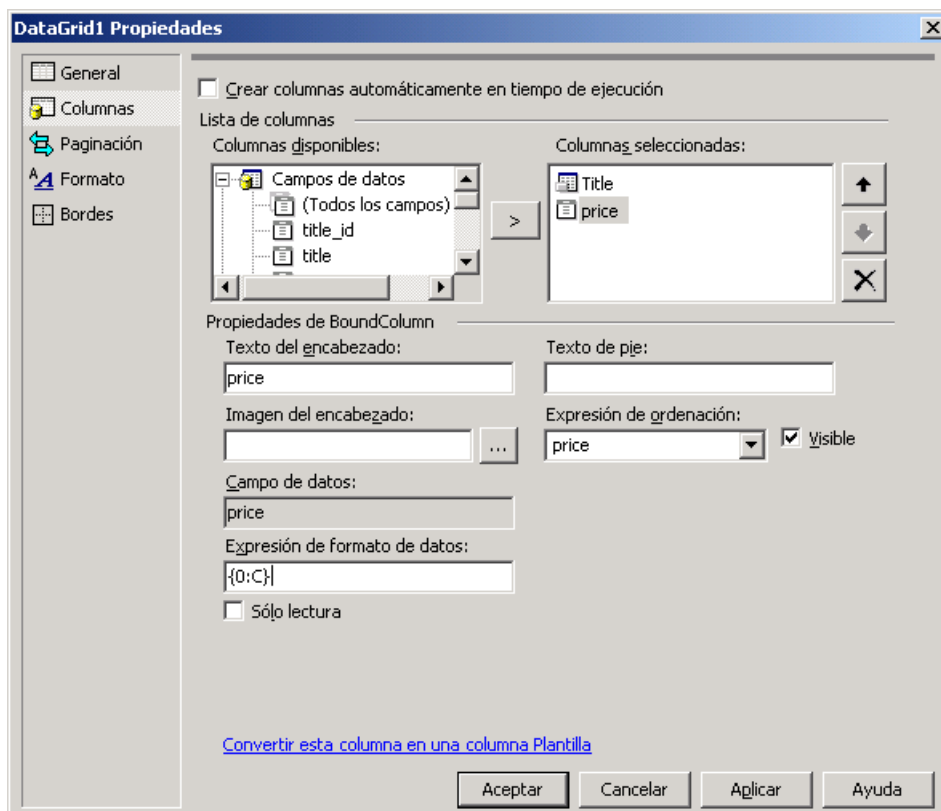
1. En el Explorador de soluciones, haga doble clic en Webform1.aspx para cambiar a la vista Diseño de la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un control **DataGrid** desde la ficha **Web Forms** del Cuadro de herramientas hasta la superficie del Diseñador de Web Forms.
4. Haga clic con el botón secundario del *mouse* en el control **DataGrid** y seleccione **Generador de propiedades** en el menú contextual.

Aparece el cuadro de diálogo **Propiedades de DataGrid1**.

5. En el formulario General, establezca el valor de la propiedad **DataSource** en **myDataSet1**, **DataMember** en **titles** y el **Campo de claves de datos** en **title\_id** (la clave principal de la base de datos).
6. En el formulario **Columnas**:
  - a. Desactive la casilla de verificación **Crear columnas automáticamente en tiempo de ejecución**.
  - b. Desplácese hacia abajo en la lista **Columnas disponibles** hasta el nodo **Columna Botón**.
  - c. Expanda el nodo **Columna Botón** y agregue el botón **Seleccionar** seleccionándolo y haciendo clic en la flecha hacia la derecha que se encuentra entre ambas listas.
  - d. Escriba "Título" en el cuadro **Texto de encabezado**. Establezca la **Expresión de ordenación** en "title" y el **Campo de texto** en "title".
  - e. Agregue la columna **price** de la lista **Columnas disponibles** seleccionándola y haciendo clic en la flecha hacia la derecha que se encuentra entre ambas listas.
  - f. En el cuadro **Texto de encabezado**, escriba un encabezado apropiado, como por ejemplo "Precio".
  - g. En el cuadro de texto **Expresión de formato de datos**, escriba la cadena de formato de moneda {0:C}.
  - h. Haga clic en **Aceptar**.

**Nota** Asegúrese de que la casilla de verificación **Visible** está activada para todas las columnas.

### Propiedades de DataGrid1



En el **DataGrid** se muestran únicamente las columnas **Título** y **Precio** del **DataSet**.

7. Cambie a la vista de código de la página de formularios Web Forms. Enlace los datos con las columnas de **DataGrid**

agregando una llamada a **DataBind()** en el método **Page\_Load**. El método presentará el siguiente aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    if (!get_IsPostBack())
    {
        myComponent.FillDataSet(myDataSet1);
        DataGrid1.DataBind();    // Add this line
    }
}
```

Para obtener más detalles sobre cómo enlazar controles, vea [Acceso a datos en las páginas de formularios Web Forms](#).

### Para probar el proyecto

1. En el menú **Generar**, haga clic en **Generar solución**.
2. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Ahora puede mostrar datos de la base de datos en una página de formulario Web Forms y verla en el explorador. La columna **Títulos** está formada por hipervínculos. Más adelante se podrá hacer clic en uno de los hipervínculos y mostrar los detalles de un libro específico.

### Ver la tabla Titles en el explorador

Title	Price
<a href="#">The Busy Executive's Database Guide</a>	\$19.99
<a href="#">Cooking with Computers: Surreptitious Balance Sheets</a>	\$11.95
<a href="#">You Can Combat Computer Stress!</a>	\$2.99
<a href="#">Straight Talk About Computers</a>	\$19.99
<a href="#">Silicon Valley Gastronomic Treats</a>	\$19.99
<a href="#">The Gourmet Microwave</a>	\$2.99
<a href="#">The Psychology of Computer Cooking</a>	
<a href="#">But Is It User Friendly?</a>	\$22.95
<a href="#">Secrets of Silicon Valley</a>	\$20.00
<a href="#">Net Etiquette</a>	
<a href="#">Computer Phobic AND Non-Phobic Individuals: Behavior Variations</a>	\$21.59
<a href="#">Is Anger the Enemy?</a>	\$10.95
<a href="#">Life Without Fear</a>	\$7.00
<a href="#">Prolonged Data Deprivation: Four Case Studies</a>	\$19.99
<a href="#">Emotional Security: A New Algorithm</a>	\$7.99
<a href="#">Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean</a>	\$20.95
<a href="#">Fifty Years in Buckingham Palace Kitchens</a>	\$11.95
<a href="#">Sushi, Anyone?</a>	\$14.99

### Agregar un panel de detalles al formulario Web Forms

El panel de detalles permite ver información adicional acerca de los libros seleccionados, sin saturar el control **DataGrid** con material ajeno a él.

En esta fase se agregará un control **DataView**, que permite filtrar la tabla para mostrar únicamente la fila seleccionada.

### Para agregar un control DataView

1. Cambie a la vista Diseño de la página de formularios Web Forms.
2. En el menú **Ver**, haga clic en **Cuadro de herramientas**.
3. Arrastre un control **DataView** desde la ficha **Datos** hasta la superficie del Diseñador de Web Forms.
4. De esta forma se agrega un nuevo objeto, dataView1, a la parte no visual de la superficie del Diseñador de Web Forms.
5. Seleccione el objeto **dataView1** y muestre su página de propiedades haciendo clic en **Ventana Propiedades** en el menú **Ver**.
6. Seleccione la propiedad **Table**. Vincule la tabla haciendo clic en los títulos de la lista desplegable.

El nombre `myDataSet1.titles` aparecerá junto a la propiedad **Table**.

En este paso se agregarán controles **Label** (etiqueta) correspondientes a columnas de datos, y se enlazarán cada **Label** con el control **DataView**.

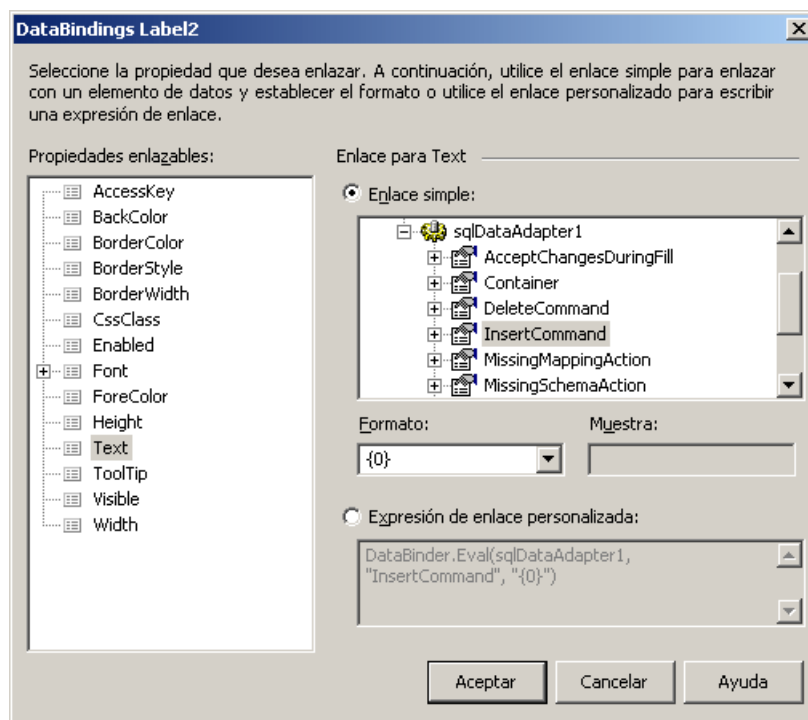
### Para agregar controles Label

1. Para cada campo de detalle utilizado (como por ejemplo **title-id**, **title**, **price** y **pubdate**), arrastre un control **Label** de la ficha **Web Forms** del Cuadro de herramientas a la superficie del Diseñador de Web Forms.
2. Seleccione cada una de las etiquetas y enlázela con un campo, de esta forma:
  - a. Abra la página **Propiedades** de la etiqueta seleccionada haciendo clic en **Ventana Propiedades** en el menú **Ver**.
  - b. Expanda la propiedad **Data**.
  - c. Seleccione la propiedad **DataBindings** haciendo clic en el botón de puntos suspensivos (...) situado junto a **DataBindings**.

Aparece el cuadro de diálogo **DataBindings** de **Label $n$**  (donde  $n$  es el número de la etiqueta). En la siguiente ilustración se muestra el cuadro de diálogo correspondiente a **Label1**.

- d. Seleccione la propiedad **Text** en **Propiedades enlazables** en el panel izquierdo, y seleccione la columna apropiada del nodo **dataView1** en el panel derecho.

### Propiedades enlazables de una etiqueta



- e. Seleccione el formato adecuado en la lista desplegable **Formato**. Por ejemplo, puede utilizar el formato **Moneda** para **price**, y uno de los formatos de fecha disponibles para **pubdate**.
  - f. Haga clic en **Aceptar**.
3. Agregue texto para describir las etiquetas, precediendo cada una de ellas con una **Etiqueta HTML** que contenga el texto descriptivo (por ejemplo, ID de título, Título, Fecha pub. Y Precio). Para ello, abra el Cuadro de herramientas y arrastre un control **Label** desde la ficha **HTML** a la superficie del Diseñador de Web Forms. Sitúe la etiqueta delante de la etiqueta relacionada correspondiente y modifique su texto de la forma apropiada. Repita este proceso para crear cuatro etiquetas.

En este paso se agregará el código necesario para activar los detalles al hacer clic en el control **DataGrid**.

### Para activar los detalles

1. Haga doble clic en el objeto **DataGrid1**.

De esta forma se agrega un controlador de eventos **DataGrid1\_SelectedIndexChanged** en el archivo de código subyacente y se cambia a la vista de código.

Modifique la definición del controlador de eventos, de la actual:



```
private void DataGrid1_SelectedIndexChanged (Object sender, System.EventArgs e)
```

a:

```
private void DataGrid1_SelectedIndexChanged (Object sender, System.EventArgs e) throws System.Data.StrongTypingException
```

2. En el método **DataGrid1\_SelectedIndexChanged**, agregue el código siguiente para configurar **dataView1.RowFilter** de forma que seleccione únicamente la fila que se desea mostrar:

```
// Visual J#  
myComponent.FillDataSet(myDataSet1);  
int index = DataGrid1.get_SelectedIndex();  
String key = DataGrid1.get_DataKeys().get_Item(index).ToString();  
dataView1.set_RowFilter(DataGrid1.get_DataKeyField ()+ "=" + key + "");
```

Conceptualmente, esto es muy parecido a especificar la cláusula WHERE de una consulta SQL y utilizarla para obtener una única fila de la tabla. La propiedad **DataKeys**, especificada al crear **DataGrid**, es un medio para identificar cada fila de forma exclusiva. El valor de la clave de la fila seleccionada se determina asignando el índice del elemento a una clave coincidente. Esta clave, que es un **title\_id** válido, se puede entonces utilizar para seleccionar de forma exclusiva la fila que se desea utilizar. Esto es así porque **title\_id** es la clave principal de la base de datos. Con otra base de datos o con una clave de datos distinta no es posible garantizar estas restricciones; por tanto, si se desea utilizar esta técnica con tablas que no disponen de una clave principal, se debe modificar el código.

3. Enlace cada una de las etiquetas utilizadas llamando a **DataBind()** con dicha etiqueta en el método **DataGrid1\_SelectedIndexChanged**. Sitúe estas llamadas después de efectuar el cambio del filtro de filas:

```
// Visual J#  
Label1.DataBind();  
Label2.DataBind();  
Label3.DataBind();  
Label4.DataBind();
```

4. Genere e inicie el proyecto.

Ahora podrá hacer clic en un título y ver los detalles con la información del nuevo título.

## Utilizar el objeto comercial externo

En este paso se utilizará la DLL escrita en Extensiones administradas de C++ (McPaymentRules.dll) para calcular el precio con descuento de un libro específico.

### Para utilizar el objeto comercial

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en el proyecto MyWebForm y haga clic en **Agregar Referencia** en el menú contextual.

Aparecerá el cuadro de diálogo **Agregar Referencia**.

2. Haga clic en **Examinar** para abrir el cuadro de diálogo **Seleccionar componente**.
3. En el cuadro de diálogo **Seleccionar componente**, haga clic en **Mis proyectos** y localice la carpeta de proyecto McPaymentRules.
4. Busque la ubicación del archivo McPaymentRules.dll en la carpeta \bin\debug, selecciónelo y haga clic en **Abrir**.

El archivo McPaymentRules.dll aparece en el panel Componentes seleccionados del cuadro de diálogo **Agregar referencia**.

5. Haga clic en **Aceptar**.
6. Arrastre un control **Label** del Cuadro de herramientas a la superficie del Diseñador de Web Forms (se le asignará el ID **Label5**).

En esta etiqueta se mostrará el precio con descuento.

7. Enlace la propiedad text de **Label5** con la columna price de **DataView1**.

No es necesario utilizar un formato específico porque dicho formato se especificará en el código.

8. Arrastre un control **HTML Label** del Cuadro de herramientas a la superficie del Diseñador de Web Forms y sitúelo delante de **Label5**.
9. Cambie el texto de la etiqueta por "Your Price".
10. Vea el archivo WebForm1.aspx.jsl y agregue el código siguiente al final del método **DataGrid1\_SelectedIndexChanged**:

```
// Visual J#
// Declare an instance of the business object:
McPaymentRules.Class1 pr = new McPaymentRules.Class1();
// Invoke the CalcDiscount Method:
try
{
    System.Decimal price = myDataSet1.get_titles().get_Item(index).get_price();
    Label5.set_Text(String.Format("{0:C}", (System.Double)pr.CalcDiscount(System.Convert.ToDouble(price))));
}
catch (Exception ex)
{
    // If the price is blank, display a message:
    Label4.set_Text(" Price is not available for this item.");
    Label5.set_Text(" Discount is not available for this item.");
}
```

11. Guárdelo todo y ejecute la aplicación.

Al hacer clic en cualquier título de la tabla, se obtienen los detalles del libro, incluido el precio con descuento (Your Price) tal como se muestra en la ilustración siguiente.

#### Información detallada acerca de un libro

Title ID:	TC7777
Title:	Sushi, Anyone?
Pub. date:	6/12/1991
Price:	\$14.99
Your Price	\$13.49

## Implementación

En este paso se implementará el proyecto en un servidor de producción específico. Para ello, copie el proyecto en el servidor de producción.

#### Para copiar el proyecto en un servidor de producción

1. Haga clic en **Copiar proyecto** en el menú **Proyecto**.

Aparece el cuadro de diálogo **Copiar proyecto**.

2. En el cuadro de texto **Carpeta de proyecto de destino**, escriba el nombre del servidor y la carpeta de proyecto; por ejemplo:

**http://ProductionServerName/MyWebForm/**

3. En la sección **Copiar**, seleccione uno de los siguientes botones de opción:

- **Sólo los archivos necesarios para ejecutar la aplicación.**
- **Todos los archivos del proyecto.**
- **Todos los archivos de la carpeta de proyecto de origen.**

4. Haga clic en **Aceptar** para iniciar el proceso de copia.

#### Cuadro de diálogo Copiar proyecto.

**Copiar proyecto**

Carpeta de proyecto de origen: http://localhost/WebApplication3

Carpeta de proyecto de destino:  
http://localhost/Copy\_of\_WebApplication3

Método de acceso Web:

☒ FrontPage

☐ Recurso compartido de archivos

Ruta de acceso:  
d:\inetpub\wwwroot\Cop\_of\_WebApplication3

Copiar:

☐ Sólo los archivos necesarios para ejecutar esta aplicación

☒ Todos los archivos de proyecto

☐ Todos los archivos de la carpeta de proyecto de origen

Aceptar Cancelar Ayuda

#### Vea también

[Extensiones administradas de C++](#) | [Servicios Web XML en Visual Studio](#) | [Tutorial: crear una aplicación distribuida con Visual J#](#) | [DataGrid \(Control Web\)](#) | [Tutoriales sobre los formularios Web Forms](#) | [Seguridad de una base de datos](#)

# Tutorial: validar los datos proporcionados por el usuario en una página de formularios Web Forms con Visual J#

Como en cualquier formulario de entrada de datos, en las páginas de formularios Web Forms es necesario comprobar que los usuarios escriben información válida, con un formato correcto, etc. Para facilitar esta tarea, puede utilizar controles de validación creados especialmente para páginas de formularios Web Forms. Los controles de validación le permiten comprobar entradas necesarias, tipos de datos, intervalos, diseños y valores específicos simplemente configurando las propiedades. Los controles realizan la comprobación automáticamente e incluyen distintas formas de mostrar la información del error a los usuarios. Para ver una introducción, vea [Validación de formularios Web Forms](#).

En este tutorial se muestra cómo crear una página de formularios Web Forms que actúe como una sencilla página de registro. La página incluye cuadros de texto para que los usuarios escriban el nombre de inicio de sesión, la fecha de nacimiento, la contraseña y la repetición de la contraseña. Se agregarán controles de validación a la página para asegurar que los usuarios rellenan todos los cuadros y que los valores tienen un formato correcto.

Para completar este tutorial, debe disponer de los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear el formulario básico mediante la creación de un proyecto y una página de formularios Web Forms y agregando los controles de entrada y las etiquetas.
- Agregar los controles de validación. Se validarán las entradas necesarias, que el usuario ha escrito un nombre de inicio de sesión y una fecha válidos y que las contraseñas coinciden.
- Pruebas.

Cuando haya finalizado el tutorial, tendrá un formulario que se parecerá al siguiente.

The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/Tutorial_validación1/WebForm1.aspx`. The browser menu bar includes 'Archivo', 'Edición', 'Ver', 'Favoritos', 'Herramientas', and 'Ayuda'. The web page contains a registration form with the following elements:

- Nombre de usuario:** A text box containing 'luis.laguna' with a red asterisk (\*) to its right.
- Fecha de nacimiento:** A text box with a red asterisk (\*) to its right.
- Contraseña:** A password box (masked with asterisks) with a red asterisk (\*) to its right.
- Repetir contraseña:** A password box (masked with asterisks) with a red asterisk (\*) to its right.
- Botón:** A button labeled 'Registrarse' at the bottom.
- Errores:** A list of red error messages on the right side:
  - Debe de incluir su fecha de nacimiento
  - El nombre debe seguir el formato nombre@dominio.xxx
  - Las contraseñas son diferentes

## Crear el formulario básico

Puede comenzar creando un formulario que contenga controles de entrada.

## Crear el formulario

El primer paso es crear un formulario al que posteriormente se le agregarán los controles de validación. Cree un nuevo proyecto de aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo `http://`, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

3. Si la página no aparece en modo cuadrícula, utilice la ventana Propiedades para establecer la propiedad **pageLayout** del objeto de documento a **GridLayout**.

Este paso es opcional, pero facilita el diseño del formulario. Es posible trabajar en diseño de flujo, pero de esta forma tendrá que hacer menos ajustes para colocar los controles en la página. La página funcionará de la misma forma en ambos casos.

## Agregar controles de entrada

En este tutorial, cuyo objetivo es crear una página de registro sencilla, agregará controles para que los usuarios puedan escribir su nombre (una dirección de correo electrónico), fecha de nacimiento y contraseña. Los usuarios deben escribir dos veces la contraseña para confirmarla.

### Para agregar los controles

1. Desde la ficha **Web Forms** del Cuadro de herramientas, agregue los siguientes controles al formulario, estableciendo su correspondiente propiedad **Id** de la siguiente forma:

Control	Propósito	ID
<b>TextBox</b>	Escribir nombre de usuario	<b>txtName</b>
<b>TextBox</b>	Escribir fecha de nacimiento	<b>txtBirthdate</b>
<b>TextBox</b>	Escribir contraseña	<b>txtPassword</b>
<b>TextBox</b>	Repetir contraseña para confirmación	<b>txtPasswordAgain</b>
<b>Button</b>	Enviar formulario	<b>btnRegister</b>

Establezca la propiedad **Text** del botón en Register.

2. Establezca la propiedad **TextMode** de los dos cuadros de contraseña en **Password** para que se muestren asteriscos (\*) mientras los usuarios escriben en ellos.
3. Agregue etiquetas delante de los cuadros de texto para indicar su función. Si la página de formularios Web Forms está en modo cuadrícula, agregue controles **Label** delante de cada cuadro de texto y establezca su propiedad **Text**. Si la página está en modo flujo, simplemente escriba texto estático delante de cada cuadro.
4. Ajuste la fuente, el tamaño y el diseño de los controles como prefiera. Para obtener más información sobre el diseño, vea [Colocar elementos HTML en la vista Diseño](#).

## Agregar controles de validación

Una vez creado el formulario de entrada, puede agregar la validación. Para ello, agregue controles de validación al formulario, un control por cada prueba de validación que desea realizar. (En total, agregará siete controles de validación para cubrir todas las pruebas que se deben realizar más un control que muestre los errores). Las pruebas de validación son las siguientes:

- Campos necesarios. Para comprobar que los usuarios escriben valores en los cuatro cuadros.
- Formato del nombre. Para este tutorial, comprobará que el nombre de inicio de sesión del usuario tiene el formato típico de los nombres de correo electrónico: *nombre@dominio.xxx*. En este tutorial, no validará que el nombre que proporciona el usuario pertenece a una cuenta de correo electrónico válida.
- Formato de fecha. Validará que la fecha de nacimiento es una fecha válida. (En este tutorial, no se validará si la fecha está dentro de un intervalo concreto).
- Coincidencia de contraseña. Validará que las contraseñas que el usuario escribe en los dos cuadros de contraseña coinciden.

## Buscar campos necesarios

El objetivo es asegurarse de que los usuarios escriben valores en los cuatro cuadros de texto. Esto significa que los cuatro cuadros de texto son campos necesarios.

### Para comprobar campos necesarios

1. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **RequiredFieldValidator** al formulario, al lado

del cuadro de texto **txtName**.

2. En la ventana **Propiedades**, seleccione el control de validación y establezca las propiedades siguientes.

Propiedad	Valor
<b>ControlToValidate</b>	txtName  Indica que el elemento que valida comprobará el contenido del campo de nombre.
<b>Text</b>	* (star)  Éste es el texto que se mostrará donde se encuentre el elemento que valida. (Es muy frecuente colocar un asterisco rojo al lado de un cuadro de entrada que contiene errores).
<b>ErrorMessage</b>	Email name is required.  Este texto se mostrará en un resumen de la página.

3. Establezca las propiedades de fuente, de color y de formato del control de validación si lo desea. Estas propiedades afectan al modo en que se muestran los errores.
4. Repita los pasos 1 a 3 para los otros tres cuadros de texto, creando tres controles **RequiredFieldValidation** más con la siguiente configuración.

Control	Valores
<b>RequiredFieldValidator2</b>	<ul style="list-style-type: none"><li>• <b>ControlToValidate:</b> establecer en txtBirthdate</li><li>• <b>Text:</b> *</li><li>• <b>ErrorMessage:</b> You must enter a birth date.</li></ul>
<b>RequiredFieldValidator3</b>	<ul style="list-style-type: none"><li>• <b>ControlToValidate:</b> establecer en txtPassword</li><li>• <b>Text:</b> *</li><li>• <b>ErrorMessage:</b> You must enter a password.</li></ul>
<b>RequiredFieldValidator4</b>	<ul style="list-style-type: none"><li>• <b>ControlToValidate:</b> establecer en txtPasswordAgain</li><li>• <b>Text:</b> *</li><li>• <b>ErrorMessage:</b> Re-enter the password to confirm it.</li></ul>

## Comprobar el formato del nombre

En este tutorial, se considera que el usuario escribe una dirección de correo electrónico como nombre de registro. Aunque no puede comprobar si el nombre representa una cuenta de correo válida, sí que puede validar que el nombre cumple el formato típico de dirección de correo (*nombre@dominio.com*). Así se capturan errores de entrada que suelen cometer los usuarios, como olvidar la cadena ".com" de la dirección de correo.

Para comprobar modelos de este tipo, puede incluir una expresión regular en un control de validación. Si conoce la sintaxis de expresiones regulares, puede crear sus propias expresiones de coincidencia de modelos. Sin embargo, también puede seleccionar una expresión de una lista de expresiones regulares predefinidas, que, por ejemplo, comprueban direcciones de correo, números de teléfono y códigos postales.

Cuando agregue este control de validación, tendrá dos controles para el cuadro de nombre. El contenido del cuadro debe pasar ambas comprobaciones para ser considerado válido. Cuando existen varios elementos de validación para un mismo control, se suele preferir que cada uno muestre su propio texto (normalmente, un asterisco) en la misma ubicación, independientemente de la validación que no se cumpla. Puede especificar esto estableciendo la propiedad **Display** del control validador. Para obtener más información, vea [Diseño de mensajes de error de validación para controles de servidor Web ASP.NET](#).

### Para comprobar el formato del nombre

1. Desde la ficha **Web Forms** del **Cuadro de herramientas**, arrastre un control **RegularExpressionValidator** al lado del cuadro **txtName**.
2. En la ventana **Propiedades**, seleccione el control de validación y establezca las propiedades siguientes.

Propiedad	Valor
<b>ControlToValidate</b>	txtName

<b>Text</b>	* (star)
<b>ErrorMessage</b>	Name must be in the format name@domain.xxx.
<b>ValidationExpression</b>	Seleccione <b>Dirección de correo electrónico de Internet</b> en la lista. Observe que de este modo establece el valor de la propiedad en una expresión regular.

## Comprobar el formato de la fecha

El objetivo es comprobar que los usuarios escriben una fecha válida en el campo de fecha de nacimiento. Para ello, se utiliza un control de validación que realiza dos funciones al mismo tiempo: comprueba que el usuario ha escrito una fecha (mediante el tipo de dato) y que ésta es posterior a una fecha especificada.

### Para comprobar el formato de la fecha

- Desde la ficha **Web Forms** del **Cuadro de herramientas**, arrastre un control **CompareValidator** al lado del cuadro **txtBirthdate**.
- En la ventana **Propiedades**, seleccione el control de validación y establezca las propiedades siguientes.

Propiedad	Valor
<b>ControlToValidate</b>	txtBirthdate
<b>Text</b>	* (star)
<b>ErrorMessage</b>	Birth date is not a valid date.
<b>ValueToCompare</b>	01/01/1900 (O cualquier otra fecha mínima)
<b>Operator</b>	Greater Than
<b>Type</b>	Date
<b>Display</b>	<b>Dynamic</b>

## Comprobar la coincidencia de contraseñas

El formulario de registro le pide a los usuarios que escriban la contraseña dos veces para confirmarla. Puede hacer esta comprobación utilizando el mismo control de validación que para la fecha, un control de comparación, pero en lugar de comparar con un valor específico, se comparará el valor de un campo, el segundo cuadro de contraseña, con el valor escrito en el primer cuadro de contraseña.

### Para comprobar la coincidencia de contraseñas

- Desde la ficha **Web Forms** del **Cuadro de herramientas**, arrastre un control **CompareValidator** al lado del cuadro **txtPasswordAgain**.
- En la ventana **Propiedades**, seleccione el control de validación y establezca las propiedades siguientes.

Propiedad	Valor
<b>ControlToValidate</b>	txtPasswordAgain
<b>Text</b>	* (star)
<b>ErrorMessage</b>	The passwords do not match.
<b>ControlToCompare</b>	txtPassword
<b>Operator</b>	Equal
<b>Display</b>	<b>Dynamic</b>

## Mostrar errores de validación

Queda un paso. Ha configurado los controles de validación para que muestren un asterisco rojo al lado de un cuadro de entrada que contenga un error. Sin embargo, puede que no sea suficiente información para el usuario. Por ejemplo, éste puede darse cuenta de que ha escrito un nombre de registro incorrecto pero no saber cuál es el error.

Para mostrar detalles sobre errores de validación, puede utilizar un control de resumen. El control muestra el valor de la propiedad **ErrorMessage** de cualquier control de validación que haya detectado un error. Si existe más de un error, el control de resumen puede mostrar todos ellos, bien en una lista de elementos con viñetas o bien como un párrafo de texto.

### Para mostrar errores de validación

- Desde la ficha **Web Forms** del **Cuadro de herramientas**, arrastre un control **ValidationSummary** al formulario. Coloque el control en un lugar donde haya espacio suficiente para mostrar varias líneas de texto.

2. Si desea mostrar los errores en un formato distinto a elementos con viñetas, seleccione una opción diferente en la propiedad **DisplayMode**.
3. Configure las propiedades **Font** y **ForeColor** para dar formato al texto del mensaje de error.
4. Ya ha terminado de agregar los controles de validación. Ahora puede hacer una prueba para ver qué tal funcionan.

## Probar la validación

El objetivo es probar la página de formularios Web Forms con valores diferentes para ver el efecto de los controles de validación.

### Para probar los controles de validación

1. En el Explorador de soluciones, haga clic con el botón secundario en el nombre del formulario y elija **Ver en el explorador**.
2. Cuando se muestre la página, escriba varios valores en los cuadros. Para comprobar la fecha de nacimiento, escriba una fecha incorrecta para la configuración regional del servidor Web.

Cada vez que cometa un error, debería aparecer un asterisco al lado del cuadro que contiene el error.

3. Haga clic en **Registrar**.

Si existe algún error, aparecerá el texto del mensaje de error en el control de resumen. (Los asteriscos rojos aparecen y desaparecen mientras escribe en los campos).

**Nota** Los campos necesarios se comprueban la primera vez que envía el formulario. Así los usuarios no se ven obligados a escribir los datos en el formulario en un orden específico. No obstante, después de enviar el formulario por primera vez, se notificarán los errores hasta que se rellenen los campos correspondientes.

Si no aparecen los errores esperados, revise la configuración de cada control de validación y pruebe de nuevo.

## Pasos siguientes

La validación que agregé a la página de formularios Web Forms en este tutorial explica los conceptos básicos de los controles de validación Web. Puede realizar más acciones con los controles de validación, tanto con su funcionalidad básica como mediante código. Puede ampliar el tutorial de las siguientes formas:

- Compruebe que el usuario escribe una fecha de nacimiento anterior al día de hoy agregando un segundo control **CompareValidator** y estableciendo su propiedad **Operator** a **LessThan** y el valor de comparación a hoy. Puede utilizar una expresión de enlace de datos para establecer el valor. Aunque no enlaza a los datos el control que valida, puede beneficiarse del hecho de que las expresiones de enlace de datos se evalúan antes de abandonar el control. La sintaxis de ASP.NET para el control puede presentar el siguiente aspecto:

```
<asp:CompareValidator id=c1
ValueToCompare='<%# System.DateTime.get_Today().ToShortDateString()
%>'
ControlToValidate="txtBirthDate" Operator="LessThan" Type="Date" Text="*" runat="server"
/>
```

**Nota** Debe llamar al método **DataBind** del control que valida para resolver la expresión.

- Compruebe que la longitud de la contraseña tiene un número mínimo de caracteres. Para ello, debe utilizar un control **RegularExpressionValidator**. No existen expresiones predefinidas que comprueben un número mínimo de caracteres. Puede utilizar una expresión como la siguiente, que obliga a escribir un mínimo de cinco caracteres:

```
^[\\w]{5,}
```

- Pruebe la validación mediante código. Cada control de validación establece una marca que puede probar mediante programación. Esto complementa la información visual que proporcionan los controles; al probar mediante código si una entrada es o no correcta, puede determinar cómo debe continuar (o no continuar) la aplicación. Para obtener información detallada, vea [Validar mediante programación los controles de servidor ASP.NET](#).

## Vea también

[Introducción a la validación de los datos proporcionados por el usuario en formularios Web Forms](#) | [Tutoriales sobre los formularios Web Forms](#) | [Seguridad de una base de datos](#)



# Tutorial: mostrar datos en una página de formularios Web Forms con Visual J#

Las páginas de formularios Web Forms proporcionan una gran variedad de maneras para trabajar con datos. En este tutorial se muestra la manera más sencilla.

En el tutorial, se mostrará información de la tabla Categories de Northwind en una cuadrícula de datos de la página. Creará un conjunto de datos, que son datos almacenados en memoria caché, que leerá información de la base de datos. Después enlazará un control [DataGrid](#) de servidor Web al conjunto de datos para mostrar los datos.

**Nota** La utilización de un conjunto de datos es sólo una de las opciones para el acceso a datos en los formularios Web Forms, y no es la manera más óptima en algunos escenarios. En este tutorial utilizará un conjunto de datos por cuestiones de simplicidad. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos Web](#).

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Northwind de SQL Server.
- Los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear un proyecto de aplicación Web y una página de formularios Web Forms.
- Crear y configurar el conjunto de datos al que se enlazará la cuadrícula. Esto incluye crear una consulta que llene el conjunto de datos a partir de la base de datos.
- Agregar el control **DataGrid** al formulario y enlazarlo a los datos.
- Agregar código para llenar el conjunto de datos.

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo http://, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Crear y configurar un conjunto de datos

En este tutorial, agregará un conjunto de datos a la página. El conjunto de datos contendrá una única tabla, la tabla Categories. (Un conjunto de datos puede contener múltiples tablas, pero en este tutorial sólo tendrá una).

El conjunto de datos no existe aún. En lugar de agregarlo manualmente al formulario, seguirá los pasos siguientes:

- Crear un adaptador de datos con un asistente. El adaptador contiene instrucciones SQL que se utilizan para leer y escribir información en la base de datos. El asistente le ayuda a definir las instrucciones SQL que necesita. Si es preciso, el asistente también crea una conexión a la base de datos.
- Generar el esquema del conjunto de datos. En este proceso, hará que Visual Studio cree una nueva clase conjunto de datos basándose en las tablas y columnas a las que está obteniendo acceso. Cuando genera la clase conjunto de datos, también agrega una instancia de ella al formulario.

Es importante que siga todos los procedimientos de esta sección. En caso contrario, la página no tendrá el conjunto de datos que se va a utilizar en las siguientes partes del tutorial.

Para obtener información detallada sobre los adaptadores de datos, vea [Introducción a los adaptadores de datos](#). Para obtener más información sobre los conjuntos de datos, vea [Introducción a los conjuntos de datos](#).

## Configurar una conexión de datos y un adaptador de datos

Para empezar, cree un adaptador de datos que contenga la instrucción SQL que se utilizará para llenar el conjunto de datos más adelante. Como parte de este proceso, defina una conexión para obtener acceso a la base de datos. Configure el adaptador de datos con el asistente, lo que facilita la creación de las instrucciones SQL necesarias para obtener acceso a los datos.

**Nota** Cuando el asistente haya finalizado, debe continuar en la sección siguiente para generar un conjunto de datos y completar la parte de acceso a datos del formulario.

### Para crear la conexión de datos y el adaptador de datos

1. Desde la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **OleDbDataAdapter** al formulario.

**Nota** También es posible usar **SqlDataAdapter**, que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa **OleDbDataAdapter** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB.

Se inicia el **Asistente para la configuración del adaptador de datos**, que ayuda a crear la conexión y el adaptador.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Northwind de SQL Server.

**Nota** Necesita permisos de lectura/escritura en el servidor de SQL Server que esté utilizando. Se recomienda especificar la seguridad integrada de Windows cuando se crea la conexión. Alternativamente, puede especificar un nombre de usuario y una contraseña y guardar dicha información con la conexión, pero eso puede afectar a la seguridad. Para obtener más información, vea [Acceso a SQL Server desde una aplicación Web](#).

- b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
- c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT CategoryID, CategoryName, Description
FROM Categories
```

Para obtener ayuda para generar la instrucción SQL, haga clic en **Generador de consultas** para iniciar el cuadro de diálogo **Generador de consultas**.

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Categories. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

- d. Haga clic en **Finalizar**.

El asistente crea una conexión (**OleDbConnection1**) que contiene información sobre cómo tener acceso a la base de datos. También tendrá un adaptador de datos (**OleDbDataAdapter1**) que contiene una consulta que define la tabla y las columnas de la base de datos a la que desea tener acceso.

3. Cuando el asistente haya finalizado, genere el conjunto de datos basándose en la consulta SQL que ha creado durante este procedimiento. Para obtener información detallada, vea la siguiente sección.

**Nota** Para obtener más información, haga clic en **Ayuda** mientras ejecuta el asistente o vea [Asistente para la configuración de adaptadores de datos](#). Para obtener información detallada sobre los adaptadores de datos, vea [Introducción a los adaptadores de datos](#).

## Crear el conjunto de datos

Después de haber establecido los métodos para conectarse a la base de datos y de especificar la información que desea (a través

del comando SQL del adaptador de datos), puede hacer que Visual Studio cree un conjunto de datos. Visual Studio puede generar el conjunto de datos automáticamente basándose en la consulta que ha especificado para el adaptador de datos. El conjunto de datos es una instancia de la clase **DataSet** basada en un esquema correspondiente (.xsd file) que describe los elementos de la clase (tabla, columnas y restricciones). Para obtener información detallada acerca de la relación entre los esquemas y los conjuntos de datos, vea [Introducción al acceso a datos con ADO.NET](#).

### Para generar un conjunto de datos

1. Desde el menú **Datos**, elija **Generar conjunto de datos**.

**Sugerencia** Si no ve el menú **Datos**, haga clic en el formulario; el formulario debe tener el foco para que aparezca el menú.

Aparecerá el cuadro de diálogo **Generar conjunto de datos**.

2. Seleccione la opción **Nuevo** y el nombre del conjunto de datos **dsCategories**.

La tabla Categories debería estar seleccionada en la lista que aparece debajo de **Elegir las tablas que desea agregar al conjunto de datos**.

3. Active la opción **Agregar este conjunto de datos al diseñador** y, a continuación, haga clic en **Aceptar**.

Visual Studio genera una clase de conjunto de datos con tipo (**dsCategories**) y un esquema que define el conjunto de datos. Verá el nuevo esquema (**dsCategories.xsd**) en el Explorador de soluciones.

**Sugerencia** En el Explorador de soluciones, haga clic en el botón **Mostrar todos los archivos** de la barra de herramientas para ver el archivo .jsl dependiente del archivo de esquema, que contiene el código que define la nueva clase de conjunto de datos.

Por último, Visual Studio agrega una instancia de la nueva clase de conjunto de datos (**DsCategories1**) al formulario.

Llegados a este punto, tiene configurado todo lo necesario para obtener información de una base de datos y almacenarla en un conjunto de datos.

### Agregar un control DataGrid para mostrar datos

En este tutorial utilizará un control **DataGrid** de servidor Web para mostrar información de la tabla Categories. También podría utilizar un control **DataList** o **Repeater** para obtener los mismos resultados. Sin embargo, el control **DataGrid** es el más sencillo de configurar (por ejemplo, no es necesario crear ninguna plantilla), lo que le permitirá concentrarse en los aspectos de los datos del tutorial.

#### Para agregar y configurar el control

1. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **DataGrid** a la página.
2. En la parte inferior de la ventana Propiedades, seleccione el vínculo **Formato automático** y elija un formato predefinido para la cuadrícula.
3. En la propiedad **DataSource**, seleccione **DsCategories1** como origen de datos. Así se enlaza la cuadrícula al conjunto de datos como un todo.
4. En la propiedad **DataMember**, seleccione **categories**. Si el origen de datos contiene más de un objeto enlazable, puede utilizar la propiedad **DataMember** para especificar qué objeto enlazar.

La configuración de estas dos propiedades enlaza la tabla de datos Categories del conjunto de datos **DsCategories1** a la cuadrícula.

### Llenar un conjunto de datos y mostrar datos en el control DataGrid

Aunque la cuadrícula está enlazada con el conjunto de datos que creó, éste no se rellena automáticamente. Debe rellenarlo usted mismo llamando a un método de adaptador de datos. Para obtener información detallada sobre cómo llenar los conjuntos de datos, vea [Introducción a los conjuntos de datos](#).

Incluso después de llenar el conjunto de datos, el control **DataGrid** no muestra los datos automáticamente. Debe enlazar de manera explícita la cuadrícula a su origen de datos. Para obtener más información, vea [Introducción al enlace de datos en las páginas de formularios Web Forms](#).

#### Para llenar el conjunto de datos y mostrar los datos en el control DataGrid

1. Haga doble clic en la página para mostrar el archivo de clase de la página en el Editor de código.

- En el controlador de eventos **Page\_Load**, llame al método **Fill** del adaptador de datos, pasándole el conjunto de datos que desea llenar:

```
// Visual J#
oleDbDataAdapter1.Fill(DsCategories1);
```

- Llame al método **DataBind** del control **DataGrid** para enlazar el control al conjunto de datos.

No es necesario volver a llenar el conjunto de datos y enlazar la cuadrícula en cada acción de ida y vuelta. Una vez que el control **DataGrid** se ha rellenado con los datos, sus valores se conservan en el estado de vista cada vez que se envía la página. Por lo tanto, sólo es necesario llenar el conjunto de datos y enlazarlo a la cuadrícula la primera vez que se llama a la página. Puede hacer una prueba utilizando la propiedad **IsPostBack** de la página.

El controlador completo puede tener el siguiente aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    if ( !get_IsPostBack())
    {
        oleDbDataAdapter1.Fill(dsCategories1);
        DataGrid1.DataBind();
        oleDbConnection1.Close();
    }
}
```

## Pruebas

Después de agregar el código, pruebe el acceso a datos en la página de formularios Web Forms.

### Para probar la página de formularios Web Forms

- Guarde la página.
- En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página y elija **Ver en el explorador**.

Compruebe que en la cuadrícula se muestra una lista de categorías.

## Pasos siguientes

- Dar formato a la cuadrícula cambiando su color, fuente, etc.
- Mostrar sólo la información seleccionada en la cuadrícula. En muchos casos, basará la presentación en información que el usuario proporciona en tiempo de ejecución (por ejemplo, podría mostrar sólo los autores de una ciudad determinada). Para ello, cree una consulta parametrizada. Para obtener información detallada, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)
- Utilice un lector de datos en lugar de un conjunto de datos como origen de datos para la cuadrícula. En una página como ésta, donde los datos son de sólo lectura, suele ser más eficiente obtener los datos directamente de la base de datos en vez de crear primero un conjunto de datos y luego llenarlo. Para ver un ejemplo, vea [Tutorial: crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J#](#).
- Separar el acceso a datos de la interfaz de usuario. En este tutorial, ha creado un formulario que tiene acceso a datos de una forma más o menos directa (a través del conjunto de datos). Un diseño más flexible y fácil de mantener es crear un componente de acceso a datos que controle el acceso a datos. El formulario (es decir, la interfaz de usuario) podría interactuar con el componente según fuera necesario. Varios formularios (y otros componentes) podrían utilizar el mismo componente, lo que elimina la sobrecarga y la redundancia de nuevos diseños de acceso a datos para cada formulario que se cree. Para obtener más información sobre la creación de accesos a datos basados en los componentes, vea [Tutorial: crear una aplicación distribuida con Visual J#](#).

## Vea también

[Acceso a datos en las páginas de formularios Web Forms](#) | [Introducción a conjuntos de datos](#) |



# Tutorial: crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J#

En aplicaciones Web, es muy frecuente que una página sólo muestre datos, es decir, que los datos de la página sean de sólo lectura. Ejemplos típicos son listas de catálogos, resultados de búsquedas, etc. Los usuarios pueden actuar en los datos haciendo clic en un botón para agregar un elemento a un carro de la compra o haciendo clic en un vínculo de la página, pero dichas acciones no afectan directamente a los datos de la página.

En este tipo de escenario, el rendimiento del acceso a datos se puede optimizar de dos formas. Una es que la página de formularios Web Forms no necesite crear y llenar un conjunto de datos. Si la página sólo va a mostrar los datos, no es necesario almacenarlos en un conjunto de datos. Por esta razón, una segunda optimización es reducir al mínimo el número de componentes de datos de la página: al no necesitar un conjunto de datos, tampoco necesitará un adaptador de datos para llenarlo. De hecho, los componentes de acceso a datos se pueden reducir a una conexión y un comando de datos que contenga una instrucción SQL o un procedimiento almacenado que se pueda ejecutar para buscar datos.

La única pieza restante es un componente de datos especial, el lector de datos, que puede buscar registros del conjunto de resultados y pasarlos a otro componente. El lector de datos está diseñado específicamente para que sea un lector de sólo lectura y sólo avance, por lo que está optimizado para obtener acceso a los datos de manera mucho más rápida. El lector de datos devuelve una estructura a la que puede enlazar controles de lista (controles **Repeater**, **DataList** y **DataGrid** de servidor Web).

Este tutorial explica cómo crear una página de formularios Web Forms con un control de servidor **DataGrid** de ASP.NET que utiliza un lector de datos para mostrar datos de sólo lectura. Cuando termine, la página tendrá el siguiente aspecto:

Página de inicio	WebForm1.aspx	WebForm1.aspx.jsl	<b>Examinar - WebForm1</b>
------------------	---------------	-------------------	----------------------------

au_id	au_fname	au_lname	phone
172-32-1176	Johnson	White	408 496-7223
213-46-8915	Marjorie	Green	415 986-7020
238-95-7766	Cheryl	Carson	415 548-7723
267-41-2394	Michael	O'Leary	408 286-2428
274-80-9391	Dean	Straight	415 834-2919
341-22-1782	Meander	Smith	913 843-0462
409-56-7008	Abraham	Bennet	415 658-9932
427-17-2319	Ann	Dull	415 836-7128
472-27-2349	Burt	Gringlesby	707 938-6445
486-29-1786	Charlene	Locksley	415 585-4620
527-72-3246	Morningstar	Greene	615 297-2723
648-92-1872	Reginald	Blotchet-Halls	503 745-6402
672-71-3249	Akiko	Yokomoto	415 935-4228
712-45-1867	Innes	del Castillo	615 996-8275
722-51-5454	Michel	DeFrance	219 547-9982
724-08-9931	Dirk	Stringer	415 843-2991
724-80-9391	Stearns	MacFeather	415 354-7128
756-30-7391	Livia	Karsen	415 534-9219

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Pubs de SQL Server.
- Los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear una página de formularios Web Forms.
- Agregar los componentes de datos necesarios.
- Agregar el control **DataGrid** para mostrar los datos y establecer algunas de sus propiedades.
- Agregar las líneas de código necesarias para buscar datos y mostrarlos en el control.

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo http://, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Agregar los componentes de datos

Para crear un acceso a los datos de sólo lectura, necesita una conexión a la base de datos de la que desea leer. También es necesario un comando de datos. El comando de datos incluye una instrucción SQL que puede ejecutar para la búsqueda de datos. (El comando de datos también puede hacer referencia a un procedimiento almacenado, pero en este tutorial se crea una sencilla instrucción SELECT de SQL).

## Crear y configurar la conexión

Para comenzar, agregue una conexión que señale a la base de datos de la que desea leer.

### Para crear la conexión de datos

1. Desde la ficha **Datos** del **Cuadro de herramientas**, arrastre un objeto **OleDbConnection** a la página.

**Nota** También es posible utilizar el objeto **SqlConnection**, que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa el comando **OleDbConnection** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB.

2. Seleccione la conexión y, en la ventana **Propiedades**, haga clic en el botón en el cuadro de propiedades **ConnectionString**.

La lista muestra todas las conexiones existentes y la opción **<Nueva conexión...>**.

3. Si aún no tiene una conexión a la base de datos Pubs de SQL Server, seleccione **<Nueva conexión...>**.

Se mostrará el cuadro de diálogo **Propiedades de vínculo de datos**.

4. Cree una conexión que señale a SQL Server y a la base de datos Pubs y, a continuación, haga clic en **Aceptar**.

**Sugerencia** Para obtener ayuda a la hora de crear una conexión, presione F1 en el cuadro de diálogo **Propiedades de vínculo de datos**.

**Nota** Necesita permisos de lectura/escritura en el servidor de SQL Server que esté utilizando. Se recomienda especificar la seguridad integrada de Windows cuando se crea la conexión. Alternativamente, puede especificar un nombre de usuario y una contraseña y guardar dicha información con la conexión, pero eso puede afectar a la seguridad. Para obtener más información, vea [Acceso a SQL Server desde una aplicación Web](#).

La información de conexión especificada se guarda como cadena de conexión de la conexión.

Esto es todo lo que necesita para realizar la conexión. Ahora debe agregar un comando con una instrucción SQL para la búsqueda de datos.

### Para crear el comando de datos y la instrucción SQL

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre un objeto **OleDbCommand** a la página.

**Nota** También es posible usar **SqlCommand**, que está optimizado para trabajar con SQL Server 7.0 o posterior. En este tutorial, se usa el comando **OleDbCommand** porque es más genérico y proporciona acceso mediante ADO.NET a cualquier origen de datos compatible con OLE DB.

2. Seleccione el comando y, en la ventana Propiedades, haga clic en el botón de la propiedad **Conexiones** y elija la conexión

realizada anteriormente. (Deberá abrir el nodo **Existentes**).

3. En la propiedad **CommandText**, haga clic en el botón de puntos suspensivos para abrir el cuadro de diálogo **Generador de consultas**.
4. Utilice el cuadro de diálogo **Generador de consultas** para construir la siguiente instrucción SQL:

```
Select au_id, au_fname, au_lname, phone  
From authors
```

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Authors. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea

[Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

5. Cuando termine, cierre el cuadro de diálogo **Generador de consultas**.

La instrucción SQL rellena la propiedad **CommandText**.

Ha terminado de agregar componentes de datos a la página. Ahora puede agregar un control para mostrar los datos.

## Agregar un control para mostrar datos

En este tutorial utilizará un control **DataGrid** de servidor Web para mostrar información de la tabla Authors. También podría utilizar un control **DataList** o **Repeater** para obtener los mismos resultados. Sin embargo, el control **DataGrid** es el más sencillo de configurar (por ejemplo, no es necesario crear ninguna plantilla), lo que le permitirá concentrarse en los aspectos de los datos del tutorial.

### Para agregar y configurar el control

1. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **DataGrid** a la página.
2. En la parte inferior de la ventana Propiedades, seleccione el vínculo **Formato automático** y elija un formato predefinido para la cuadrícula.

Si ya ha trabajado antes con un control **DataGrid** enlazado a datos, quizás espere tener que establecer la propiedad **DataSource** del control. Sin embargo, en este ejemplo no establecerá aquí el origen de datos; lo establecerá utilizando código que señale al lector de datos.

También observará que las columnas de la cuadrícula no se establecerán explícitamente. En este ejemplo, los enlaces de datos para la cuadrícula se establecen en tiempo de ejecución, y la cuadrícula creará y mostrará las columnas dinámicamente basándose en los datos a los que está enlazada.

**Nota** Si está familiarizado con el control **DataGrid**, puede dar un formato previo a las cuatro columnas (Identificador, Nombre, Apellidos y Teléfono). Como este paso no es necesario (la cuadrícula creará las columnas automáticamente en tiempo de ejecución cuando establezca el enlace), el tutorial no muestra los pasos de este procedimiento.

## Agregar código para buscar y mostrar los datos

Para crear el acceso optimizado de sólo lectura a datos, ejecutará la instrucción SQL creada con el comando de datos. El comando devuelve un lector de datos al que posteriormente puede enlazarse. Puede realizar todo esto con unas simples líneas de código.

### Para agregar código para buscar datos

1. Haga doble clic en la página de formularios Web Forms para abrir el Editor de código y buscar el controlador de eventos **Page\_Load**.
2. Cree el código que realice las siguientes funciones:
  - a. Crear una variable de instancia de tipo **OleDbDataReader**.
  - b. Abrir la conexión.
  - c. Llamar al método **ExecuteReader** del comando de datos, que ejecuta la instrucción SQL del comando de datos y devuelve el conjunto de resultados en el objeto del lector de datos.
  - d. Establecer el origen de datos del control **DataGrid** para señalar al lector de datos. Este paso no se podía realizar antes en la ventana Propiedades porque todavía no existía el lector de datos.
  - e. Enlazar el control **DataGrid** al lector de datos.
  - f. Cerrar el lector de datos.



- g. Cerrar la conexión. Este es un paso importante; la conexión permanece abierta mientras se están obteniendo los datos de la base de datos. Por lo tanto, cuando ya no necesite la conexión, es mejor cerrarla para liberar los recursos que utiliza.

En el siguiente ejemplo se muestra el código necesario para realizar los pasos anteriores:

**Nota** Por cuestiones de claridad, el código crea de manera explícita una variable que aloja el lector de datos.

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    OleDbConnection1.Open();
    System.Data.OleDb.OleDbDataReader Reader;
    Reader = OleDbCommand1.ExecuteReader();
    DataGrid1.set_DataSource ( Reader);
    DataGrid1.DataBind();
    Reader.Close();
    OleDbConnection1.Close();
}
```

## Pruebas

Después de agregar el código, pruebe el acceso a datos en la página de formularios Web Forms.

### Para probar la página de formularios Web Forms

1. Guarde la página.
2. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página y elija **Ver en el explorador**.

Compruebe que en la cuadrícula se muestra una lista de autores.

## Pasos siguientes

En este tutorial se han mostrado los pasos básicos relacionados con el uso de un lector de datos para la presentación de datos de sólo lectura en la página. Algunas mejoras que podría realizar en el formulario en este tutorial incluyen:

- Dar formato a las columnas de la cuadrícula para mostrar la información disponible en tiempo de ejecución.
- Dar formato a la cuadrícula cambiando su color, fuente, etc.
- Mostrar los datos en otro control, como un control **DataList** o **Repeater**. El proceso es prácticamente el mismo. Sin embargo, se diferencia en que es necesario crear plantillas, arrastrar controles a las plantillas y enlazar los controles. Debido a que el enlace de datos del control contenedor se realiza en tiempo de ejecución, debe crear en las plantillas expresiones de enlace personalizado para los controles. La expresión de enlace personalizado típica es la siguiente:

```
DataBinder.Eval(Container, "DataItem.au_lname")
```

Sólo variaría la parte final de la expresión (`au_lname`) para cada control de la plantilla.

Para obtener más información sobre las plantillas, vea [Plantillas de controles de servidor Web](#).

- Utilizar una consulta parametrizada para obtener datos, que es un escenario muy común en la realidad. En ese caso, se crearía una instrucción SQL que aceptase parámetros y, a continuación, se pasarían dichos parámetros rellenando la colección **Parameters** de un comando de datos antes de ejecutarlo. Para obtener más información, vea [Establecer y obtener parámetros de comandos de datos](#).
- Agregar paginación. Ésta es una variación de una consulta parametrizada. Es necesario mantener en la página alguna indicación de qué registros se han mostrado. Cuando el usuario se desplaza a la página siguiente, es necesario pasar la información del parámetro con la consulta para obtener el valor de los datos de la siguiente página. Para obtener información detallada sobre la creación de resultados paginados, vea [Comportamiento de la paginación en los controles DataGrid de servidor Web](#).

## Vea también



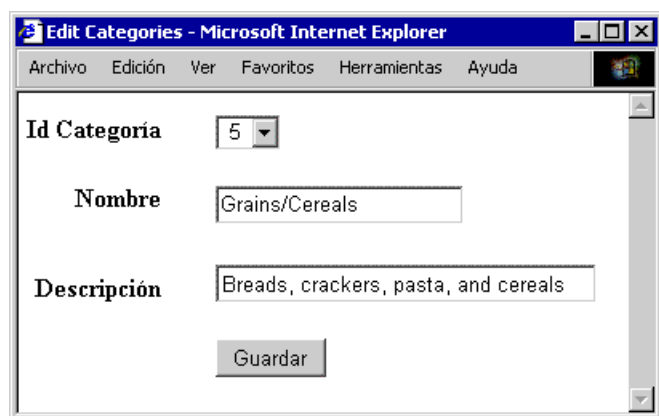
# Tutorial: actualizar datos mediante una consulta de actualización de bases de datos en los formularios Web Forms con Visual J#

La arquitectura de enlace de datos de los formularios Web Forms le permite enlazar las propiedades del control a un origen de datos, por lo que es muy sencillo mostrar los datos en la página. Sin embargo, la arquitectura de enlace de datos no admite actualizaciones, es decir, que en las páginas de formularios Web Forms el enlace es en un sólo sentido. Como la mayoría de las aplicaciones Web implican la lectura de datos pero no su actualización, el enlace en un sólo sentido permite que el tamaño y la complejidad de la arquitectura de la página sean los mínimos en los escenarios más comunes.

Sin embargo, habrá ocasiones en las que necesitará actualizar datos. Existen varias formas de lograr esto. Si está utilizando un conjunto de datos, puede realizar cambios en el conjunto de datos y después llamar al adaptador de datos para que envíe los cambios desde el conjunto de datos a la base de datos. Para obtener un ejemplo de este escenario, vea [Tutorial: utilizar un control Web DataGrid para leer y escribir datos con Visual J#](#). Alternativamente, puede actualizar la base de datos directamente, utilizando instrucciones SQL o procedimientos almacenados, que es lo que muestra este tutorial.

**Nota** Para obtener información detallada a la hora de elegir entre estos enfoques, vea [Recomendaciones de la estrategia de acceso a datos en el Web](#).

En este tutorial se muestra cómo crear una página de formularios Web Forms que lee y escribe datos de una base de datos mediante instrucciones SQL. En este tutorial, trabajará con la tabla Categories de la base de datos Northwind de SQL Server. En la página, puede seleccionar un registro, modificarlo y guardarlo en la base de datos. Cuando termine, la página tendrá el siguiente aspecto:



En el tutorial se explicarán los siguientes conceptos:

- Crear objetos de comandos que incorporan instrucciones SQL.
- Utilizar una instrucción SQL y un lector de datos para obtener datos de la base de datos.
- Crear consultas parametrizadas.
- Configurar parámetros de consulta y ejecutar consultas.

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Northwind de SQL Server.
- Los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear una página de formularios Web Forms.
- Agregar los componentes de datos necesarios.
- Agregar controles para mostrar los datos.
- Agregar código para leer y escribir datos.

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo http://, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Agregar los componentes de datos

Para tener acceso a los datos, necesita una conexión a la base de datos de la que desea leer. También necesitará varios objetos de comandos: dos para leer datos de la base de datos utilizando instrucciones Select de SQL y otro para actualizar la base de datos con una instrucción Update de SQL. Dos de estos comandos necesitarán parámetros para poder pasar valores en tiempo de ejecución a las instrucciones antes de ejecutarlas. (Los comandos de datos también puede hacer referencia a un procedimiento almacenado en lugar de a instrucciones SQL, pero en este tutorial se utilizarán instrucciones).

## Crear y configurar la conexión

Para comenzar, agregue una conexión que señale a la base de datos de la que desea leer.

### Para crear la conexión de datos

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre un objeto **SqlConnection** hasta la superficie del Diseñador.

**Nota** Como está utilizando SQL Server, se utiliza el objeto **SqlConnection**, que está optimizado para trabajar con SQL Server 7.0 o posterior. Si está utilizando una base de datos diferente, utilizará el objeto **OleDbConnection**, que proporciona acceso ADO.NET a cualquier origen de datos compatible con OLE DB.

2. Seleccione la conexión y, en la ventana **Propiedades**, haga clic en el botón en el cuadro de propiedades **ConnectionString**.

La lista muestra todas las conexiones existentes y la opción **<Nueva conexión...>**.

3. Si aún no tiene una conexión a la base de datos Northwind de SQL Server, seleccione **<Nueva conexión...>**.

Se mostrará el cuadro de diálogo **Propiedades de vínculo de datos**.

4. Cree una conexión que señale a SQL Server y a la base de datos Northwind y, a continuación, haga clic en **Aceptar**.

**Nota** Necesita permisos de lectura/escritura en el servidor de SQL Server que esté utilizando. Se recomienda especificar la seguridad integrada de Windows cuando se crea la conexión. Alternativamente, puede especificar un nombre de usuario y una contraseña y guardar dicha información con la conexión, pero eso puede afectar a la seguridad. Para obtener más información, vea [Acceso a SQL Server desde una aplicación Web](#).

La información de conexión especificada se guarda como cadena de conexión de la conexión.

Ahora es necesario agregar comandos. Necesitará tres en total, cada uno con una consulta diferente:

- Uno para obtener todos los identificadores de categoría.
- Otro para obtener la categoría seleccionada basándose en un identificador proporcionado.
- Otro para actualizar una categoría con nuevos valores.

### Para crear un comando de datos para obtener todos los identificadores de categorías

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre un objeto **SqlCommand** hasta la superficie del Diseñador.
2. Cambie el nombre del comando a **cmdCategoriesAll**.
3. Seleccione el comando y, en la ventana Propiedades, haga clic en el botón de la propiedad **Conexiones** y elija la conexión realizada anteriormente. (Deberá abrir el nodo **Existentes**).

- En la propiedad **CommandText**, haga clic en el botón de puntos suspensivos para abrir el cuadro de diálogo **Generador de consultas**.
- Utilice el cuadro de diálogo **Generador de consultas** para construir la siguiente instrucción SQL:

```
SELECT CategoryID, CategoryName, Description FROM Categories
```

- Cuando termine, cierre el cuadro de diálogo **Generador de consultas**.

La instrucción SQL rellena la propiedad **CommandText**.

El resto de comandos son más complejos, ya que requieren parámetros.

#### Para crear un comando de datos para obtener un único registro de categoría

- Arrastre un segundo objeto **SqlCommand** hasta la superficie del Diseñador y establezca su propiedad **Connection** en **SqlConnection1**.
- Llame al comando **cmdCategoriesById**.
- En la propiedad **CommandText**, haga clic en el botón de puntos suspensivos para abrir el cuadro de diálogo **Generador de consultas**.
- Utilice el cuadro de diálogo **Generador de consultas** para construir la siguiente instrucción SQL:

```
SELECT CategoryID, CategoryName, Description
FROM Categories
WHERE (CategoryID = @categoryid)
```

**Nota** La variable **@categoryid** es una "variable con nombre". El nombre exacto de la variable que utiliza en la cláusula Where no tiene importancia, pero necesitará conocerlo posteriormente en el tutorial. El prefijo "@" es necesario para los parámetros con nombre de SQL Server.

El último comando incluye una instrucción **Update** para enviar los cambios a la base de datos.

#### Para crear un comando de datos para actualizar un registro de categoría

- Arrastre un tercer objeto **SqlCommand** hasta la superficie del Diseñador y establezca su propiedad **Connection** en **SqlConnection1**.
- Llame al comando **cmdCategoriesUpdate**.
- Establezca la propiedad **CommandText** del comando en la siguiente instrucción:

```
UPDATE Categories
SET CategoryName = @categoryname, Description = @categorydescription WHERE (CategoryID = @categoryid)
```

Ha terminado de agregar componentes de datos a la página.

## Agregar controles

En esta página, necesitará una lista desplegable para permitir a los usuarios que seleccionen un identificador de categoría, varios cuadros de texto que permitan editar texto y un botón para guardar los cambios.

#### Para agregar controles al formulario

- Desde la ficha **Web Forms** del Cuadro de herramientas, agregue los siguientes controles al formulario, estableciendo las propiedades especificadas de la siguiente forma:

Control	Propósito	Propiedades
<b>DropDownList</b>	Permite a los usuarios seleccionar una lista de categorías disponible en tiempo de ejecución.	ID: <b>ddlCategoryID</b> AutoPostBack: <b>true</b>
<b>TextBox</b>	Permite a los usuarios modificar el nombre de la categoría.	ID: <b>txtCategoryName</b>
<b>TextBox</b>	Permite a los usuarios modificar la descripción de la categoría.	ID: <b>txtCategoryDescription</b>

<b>Button</b>	Actualiza la base de datos con los cambios.	ID: <b>btnSave</b> Text: <b>Save</b>
---------------	---	---

2. Agregue etiquetas delante de los controles para indicar su función.

**Sugerencia** Si está trabajando en modo cuadrícula, puede agregar texto mediante el panel de modo flujo ([Controles Div HTML](#)) desde la ficha **HTML** del Cuadro de herramientas.

## Agregar código para mostrar y actualizar los datos

Para terminar la página, deberá agregar código para realizar las tres tareas siguientes:

- Cuando se inicialice la página, debe llenar la lista desplegable con los identificadores de categorías. Como el primer identificador de categoría se seleccionará automáticamente, también puede mostrar el nombre y la descripción de la categoría en los cuadros de texto.
- Cuando los usuarios seleccionen un identificador de categoría en la lista desplegable, debe determinar el identificador seleccionado. Después utilizará dicho valor para ejecutar la instrucción SQL que obtendrá un único registro.
- Cuando los usuarios hagan clic en el botón Guardar, debe obtener las modificaciones y utilizarlas para ejecutar la instrucción Update con los nuevos valores.

## Llenar la lista desplegable al inicializar la página

Para llenar la lista desplegable, ejecutará una consulta que devolverá todos los registros de la tabla Categories. Sólo es necesario realizar esto la primera vez que se ejecuta la página; después, los valores de la lista se conservan en el estado de vista del control.

Después de ejecutar la consulta, utilizará un lector de datos (objeto [SqlDataReader](#)) para obtener los registros. El lector de datos está diseñado específicamente para que sea un lector de sólo lectura y sólo avance, por lo que está optimizado para obtener acceso a los datos de manera mucho más rápida.

### Para agregar código para llenar la lista desplegable

1. Haga doble clic en la página de formularios Web Forms para abrir el Editor de código y buscar el controlador de eventos **Page\_Load**.
2. Cree el código que realice las siguientes funciones:
  - a. Pruebe la propiedad **IsPostBack** de la página para determinar si es la primera vez que se ejecuta. En ese caso, realice el resto de pasos.
  - b. Crear una variable de instancia de tipo **SqlDataReader**.
  - c. Abrir la conexión.
  - d. Llamar al método **ExecuteReader** del objeto **cmdCategoriesAll**, que ejecuta la instrucción SQL del comando de datos y devuelve el conjunto de resultados en el objeto del lector de datos.
  - e. Proporcionar una manera, por ejemplo, un indicador booleano, para saber cuándo se está obteniendo el primer registro, porque se mostrará sólo la información del registro.
  - f. Mediante un bucle, utilizar el lector de datos para obtener los registros sucesivos.
  - g. Para cada registro, crear un nuevo elemento en la lista desplegable y asignarle el identificador de categoría.
  - h. Sólo en el caso del primer registro, obtener el nombre y la descripción de la categoría y configurar con ellos la propiedad **Text** de los cuadros de texto.
  - i. Cerrar el lector de datos.
  - j. Cerrar la conexión. Este es un paso importante; la conexión permanece abierta mientras se están obteniendo los datos de la base de datos. Por lo tanto, cuando ya no necesite la conexión, es mejor cerrarla para liberar los recursos que utiliza.

En el siguiente ejemplo se muestra el método **Page\_Load** completo con el código necesario para realizar los pasos anteriores:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    if (!this.get_IsPostBack())
    {
        // Executes only the first time the page is processed. After
```

```

// that, the list is already in the drop-down list and is
// preserved in view state.
System.Data.SqlClient.SqlDataReader dreader;
sqlConnection1.Open();
dreader = cmdCategoriesAll.ExecuteReader();
boolean firstrow = true;
while (dreader.Read())
{
    ddlCategoryID.get_Items().Add(new ListItem(dreader.get_Item(0).ToString()));
    if (firstrow)
    {
        txtCategoryName.set_Text(dreader.get_Item(1).ToString());
        txtCategoryDescription.set_Text ( dreader.get_Item(2).ToString());
        firstrow = false;
    }
}
dreader.Close();
sqlConnection1.Close();
}
}

```

## Mostrar un registro cuando el usuario selecciona un identificador de categoría

La lista desplegable muestra los identificadores de categorías. Cuando el usuario selecciona una, es decir, cuando se provoca el evento **SelectedIndexChanged** del control, desea mostrar el registro correspondiente.

Cuando creó la lista desplegable, estableció su propiedad **AutoPostBack** en **true**. El resultado es que la página se envía al servidor tan pronto como el usuario hace una selección.

Para obtener un registro de categoría, ejecutará una instrucción SQL que utiliza una cláusula Where para encontrar sólo la categoría cuyo identificador seleccionó el usuario. Para pasar este valor a la instrucción, establece el valor de un parámetro (un objeto **SqlParameter**) que forma parte de un comando de datos.

### Para agregar código para llenar la lista desplegable

1. Haga doble clic en la lista desplegable para abrir el Editor de código y crear el controlador del evento **SelectedIndexChanged**.
2. Cree el código que realice las siguientes funciones:
  - a. Obtener el texto de la selección del usuario en la lista desplegable.
  - b. Establecer el parámetro (@categoryid) del comando **cmdCategoriesById** para el identificador seleccionado.
  - c. Abrir la conexión.
  - d. Crear una variable de instancia de lector de datos y llamar al método **ExecuteReader** del objeto **cmdCategoriesById** para ejecutar el comando. En este caso, pasará un indicador al método indicando que sólo desea devolver un registro.
  - e. Utilizar el lector de datos para obtener el registro devuelto y, a continuación, mostrar sus valores en los cuadros de texto.
  - f. Cerrar el lector de datos.
  - g. Cerrar la conexión.

En el siguiente ejemplo se muestra el método **SelectedIndexChanged** completo con el código necesario para realizar los pasos anteriores:

```

// Visual J#
private void ddlCategoryID_SelectedIndexChanged (Object sender, System.EventArgs e)
{
    String categoryid;
    categoryid = ddlCategoryID.get_SelectedItem().get_Text();
    ((System.Data.SqlClient.SqlParameterCollection)cmdCategoriesById.get_Parameters()).
        get_Item("@categoryid").set_Value(categoryid);
    sqlConnection1.Open();
    System.Data.SqlClient.SqlDataReader dreader;

```

```

dreader = cmdCategoriesById.ExecuteReader(CommandBehavior.SingleRow);
if (dreader.Read())
{
    txtCategoryName.set_Text(dreader.get_Item(1).ToString());
    txtCategoryDescription.set_Text(dreader.get_Item(2).ToString());
}
dreader.Close();
sqlConnection1.Close();
}

```

## Actualizar un registro

Por último, vamos a agregar código para actualizar la base de datos con los cambios realizados en el registro mostrado en la página. Al igual que ocurre cuando se obtiene un único registro, es necesario establecer algunos parámetros para la actualización y ejecutar una instrucción (en este caso, una instrucción Update). Sin embargo, como la instrucción Update no devuelve registros, el código para ejecutarla es relativamente sencillo.

### Para actualizar registros

1. Haga doble clic en el botón Guardar para crear un controlador **Click**.
2. Cree el código que realice las siguientes funciones:
  - a. Establezca los parámetros (tres en total) en los valores de los controles de la pantalla.
  - b. Abrir la conexión.
  - c. Llame al método **ExecuteNonQuery** del comando **cmdCategoriesUpdate**.
  - d. Cerrar la conexión.

En el siguiente ejemplo se muestra el método **Click** completo con el código necesario para realizar los pasos anteriores:

```

// Visual J#
private void btnSave_Click (Object sender, System.EventArgs e)
{
    ((System.Data.SqlClient.SqlParameterCollection) cmdCategoriesUpdate.get_Parameters()).
        get_Item("@categoryid").set_Value(ddlCategoryID.
            get_SelectedItem().get_Text());
    ((System.Data.SqlClient.SqlParameterCollection)cmdCategoriesUpdate.get_Parameters()).
        get_Item("@categoryname").set_Value(txtCategoryName.get_Text());
    ((System.Data.SqlClient.SqlParameterCollection)cmdCategoriesUpdate.get_Parameters()).
        get_Item("@categorydescription").set_Value(txtCategoryDescription.get_Text());
    sqlConnection1.Open();
    cmdCategoriesUpdate.ExecuteNonQuery();
    sqlConnection1.Close();
}

```

## Pruebas

Ahora puede comprobar el formulario para asegurarse de que muestra los datos de Categories y que los usuarios pueden realizar actualizaciones.

### Para probar la página

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página y elija **Ver en el explorador**.
2. Cuando se muestre la página, compruebe que la lista desplegable contiene una lista de identificadores de categorías y que la primera categoría se muestra en los cuadros de texto.
3. Seleccione una categoría diferente en la lista.

Se mostrará el nuevo registro.

**Nota** Si al seleccionar un nuevo identificador de categoría en la lista desplegable no cambia el registro, compruebe que la propiedad **AutoPostBack** del control está establecida en **true**.



4. Modifique el texto en uno de los cuadros de texto y, a continuación, haga clic en el botón **Guardar**.
5. Desplácese a otro registro de categoría y vuelva al que modificó anteriormente.

Los cambios realizados serán visibles cuando se vuelva a mostrar el registro.

## Pasos siguientes

En este tutorial se han explicado los pasos básicos que implica la utilización de un lector de datos para obtener registros directamente de la base de datos y de un comando de datos para realizar actualizaciones. Para ampliar la página de formularios Web Forms que ha creado, podría inspirarse en las siguientes ideas:

- Proporcionar a los usuarios una manera alternativa para seleccionar categorías. En este tutorial, la lista desplegable mostraba identificadores de categorías, que normalmente no aportan mucha información a los usuarios. Esto permitía a los usuarios modificar el nombre y mostraba claramente cómo se iba moviendo la información de los registros de la base de datos dentro y fuera de los controles.

Un cambio relativamente sencillo sería mostrar los nombres de categorías como el texto de la lista desplegable (la información que se muestra en la lista) y los identificadores de categorías como el valor de la lista. (Sin embargo, esto no permitiría que los usuarios modificaran el nombre). Después podría pasar el valor como parámetro en lugar de pasar el texto.

- Agregar comprobación de errores. Para que el código sea más sólido, se suelen encerrar todas las instrucciones de acceso a datos en bloques try-catch. Puede encontrarse con distintos problemas relacionados con el acceso a datos (muchos de ellos están fuera del control de la aplicación, como los problemas de red). Para obtener información detallada, vea [Procedimientos recomendados para iniciar excepciones desde componentes](#).
- Agregar un control de concurrencia. En este tutorial, sólo se actualiza un registro en la base de datos sin determinar primero si otro usuario ha realizado también cambios al mismo tiempo. Para obtener información detallada, vea [Control de concurrencia en ADO.NET](#).

## Vea también

[Tutorial: crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J#](#) |  
[Introducción a las herramientas de diseño de conexiones ADO.NET](#) | [Introducción a los objetos DataCommand en Visual Studio](#) |  
[Ejecutar un comando de datos que devuelve un conjunto de resultados](#) |  
[Ejecutar comandos de actualización y de base de datos utilizando un comando de datos](#) |  
[Establecer y obtener parámetros de comandos de datos](#) | [Control de concurrencia en ADO.NET](#) |  
[Tutoriales sobre los formularios Web Forms](#)

# Tutorial: utilizar un control Web DataGrid para leer y escribir datos con Visual J#

La arquitectura de enlace de datos en los formularios Web Forms le permite mostrar fácilmente datos en los controles de la página. Sin embargo, el enlace de datos no es de doble sentido, es decir, lee los datos de un origen de datos pero no los actualiza. Actualizar datos es más complejo que mostrarlos, y como la mayoría de las páginas de formularios Web Forms no necesitan que los datos se escriban de nuevo en el origen, el enlace de datos de formularios Web Forms reduce al mínimo el tamaño y procesamiento de la página al no incluir código de actualización.

Por supuesto, en ocasiones es necesario crear una página de formularios Web Forms que actualice datos. En este tutorial se muestra una manera de realizar esto. Describe cómo puede utilizar un control **DataGrid** para mostrar datos, permitir a los usuarios modificarlos y enviar los datos modificados de vuelta al origen.

Cuando ejecute la página, tendrá el siguiente aspecto:

	CategoryID	CategoryName	Description
<a href="#">Edición</a>	1	Beverages	Soft drinks, coffees, teas, beers, and ales
<a href="#">Edición</a>	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
<a href="#">Edición</a>	3	Confections	Desserts, candies, and sweet breads
<a href="#">Edición</a>	4	Dairy Products	Cheeses
<a href="#">Edición</a>	5	Grains/Cereals	Breads, crackers, pasta, and cereal
<a href="#">Edición</a>	6	Meat/Poultry	Prepared meats
<a href="#">Edición</a>	7	Produce	Dried fruit and bean curd
<a href="#">Edición</a>	8	Seafood	Seaweed and fish

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Northwind de SQL Server.
- Los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear una página de formularios Web Forms.
- Agregar los componentes de datos necesarios.
- Agregar el control **DataGrid** para mostrar los datos.
- Agregar código para leer datos de la base de datos y enlazarlos con la cuadrícula.
- Configurar el control **DataGrid** para permitir que los usuarios modifiquen datos.
- Agregar el código necesario para actualizar los datos.

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo `http://`, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada `WebForm1.aspx` en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Crear y configurar un conjunto de datos

En las páginas de formularios Web Forms, existen distintas opciones para tener acceso a los datos. Una es utilizar un conjunto de datos, que almacena los datos en memoria caché. Como alternativa, se puede tener acceso directamente a la base de datos, utilizando comandos de datos para ejecutar instrucciones SQL o procedimientos almacenados. Como regla general, es más fácil actualizar datos con un conjunto de datos, y esto es lo que se utiliza en este tutorial. Para obtener más información, vea [Recomendaciones sobre la estrategia de acceso a datos Web](#).

No va a agregar directamente un conjunto de datos a la página. En su lugar, seguirá los pasos siguientes:

1. Crear un adaptador de datos con un asistente. El adaptador contiene instrucciones SQL que se utilizan para leer y escribir información en la base de datos. El asistente le ayuda a definir las instrucciones SQL que necesita. Si es preciso, el asistente también crea una conexión a la base de datos.
2. Generar el esquema del conjunto de datos. En este proceso, hará que Visual Studio cree una nueva clase conjunto de datos basándose en las tablas y columnas a las que está obteniendo acceso. Cuando genera la clase conjunto de datos, también agrega una instancia de ella al formulario.

Es importante que siga todos los procedimientos de esta sección. En caso contrario, la página no tendrá el conjunto de datos que se va a utilizar en las siguientes partes del tutorial.

Para obtener una introducción a los adaptadores de datos, vea [Introducción a los adaptadores de datos](#). Para obtener una introducción a los conjuntos de datos, vea [Introducción a los conjuntos de datos](#).

## Configurar una conexión de datos y un adaptador de datos

Para empezar, cree un adaptador de datos que contenga la instrucción SQL que se utilizará para llenar el conjunto de datos más adelante. Como parte de este proceso, defina una conexión para obtener acceso a la base de datos. Configure el adaptador de datos con el asistente, lo que facilita la creación de las instrucciones SQL necesarias para obtener acceso a los datos.

**Nota** Cuando el asistente haya finalizado, debe continuar en la sección siguiente para generar un conjunto de datos y completar la parte de acceso a datos de la página.

### Para crear la conexión de datos y el adaptador de datos

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre un objeto **SqlDataAdapter** a la página.

**Nota** Si no está trabajando con una base de datos de SQL Server, utilizará un adaptador del tipo **OleDbDataAdapter**, que proporciona acceso a cualquier origen de datos compatible con OLE DB.

Se inicia el **Asistente para la configuración del adaptador de datos**, que ayuda a crear la conexión y el adaptador.

2. En el asistente, haga lo siguiente:
  - a. En la segunda página, cree o elija una conexión que apunte a la base de datos Northwind de SQL Server. Para obtener más información sobre cómo obtener acceso a la base de datos, consulte con el administrador de base de datos.

**Nota** Necesita permisos de lectura/escritura en el servidor de SQL Server que esté utilizando. Se recomienda especificar la seguridad integrada de Windows cuando se crea la conexión. Alternativamente, puede especificar un nombre de usuario y una contraseña y guardar dicha información con la conexión, pero eso puede afectar a la seguridad. Para obtener más información, vea [Acceso a SQL Server desde una aplicación Web](#).

- b. En la tercera página, especifique que desea usar una instrucción SQL para obtener acceso a la base de datos.
- c. En la cuarta página, cree la siguiente instrucción SQL:

```
SELECT CategoryID, CategoryName, Description
FROM Categories
```

Para obtener ayuda para generar la instrucción SQL, haga clic en **Generador de consultas** para iniciar el cuadro de diálogo **Generador de consultas**.

**Nota** En este tutorial llenará el conjunto de datos con todas las filas de la tabla Categories. En aplicaciones de producción, normalmente se optimiza el acceso a datos mediante la creación de una consulta que sólo devuelve las columnas y filas que se necesitan. Para obtener un ejemplo, vea [Tutorial: mostrar datos en un formulario Windows Forms mediante una consulta parametrizada con Visual J#](#)

- d. Haga clic en **Finalizar**.

El asistente crea una conexión (**SqlConnection1**) que contiene información sobre cómo tener acceso a la base de datos. También tendrá un adaptador de datos (**sqlDataAdapter1**) que contiene una consulta que define la tabla y las columnas de la base de datos a la que desea tener acceso.

Cuando el asistente haya finalizado, genere el conjunto de datos basándose en la consulta SQL que ha creado durante este procedimiento. Para obtener información detallada, vea la siguiente sección.

## Crear el conjunto de datos

Después de haber establecido los métodos para conectarse a la base de datos y de especificar la información que desea (a través del comando SQL del adaptador de datos), puede hacer que Visual Studio cree un conjunto de datos. Visual Studio puede generar el conjunto de datos automáticamente basándose en la consulta que ha especificado para el adaptador de datos. El conjunto de datos es una instancia de la clase **DataSet** basada en un esquema correspondiente (.xsd file) que describe los elementos de la clase (tabla, columnas y restricciones). Para obtener información detallada acerca de la relación entre los esquemas y los conjuntos de datos, vea [Introducción al acceso a datos con ADO.NET](#).

### Para generar un conjunto de datos

1. Desde el menú **Datos**, elija **Generar conjunto de datos**.

**Sugerencia** Si el comando **Generar conjunto de datos** no está habilitado, haga clic en la página; la página debe tener el foco para que aparezca el comando.

Aparecerá el cuadro de diálogo **Generar conjunto de datos**.

2. Seleccione la opción **Nuevo** y el nombre del conjunto de datos **dsCategories**.

Compruebe que la tabla **Categories** está seleccionada en la lista que aparece debajo de **Elegir las tablas que desea agregar al conjunto de datos**.

3. Asegúrese de que la casilla **Agregar este conjunto de datos al diseñador** está seleccionada y haga clic en **Aceptar**.

Visual Studio genera una clase de conjunto de datos con tipo (**dsCategories**) y un esquema que define el conjunto de datos. Verá el nuevo esquema (**dsCategories.xsd**) en el Explorador de soluciones.

**Sugerencia** En el Explorador de soluciones, haga clic en el botón **Mostrar todos los archivos** de la barra de herramientas para ver el archivo .jsl dependiente del archivo de esquema, que contiene el código que define la nueva clase de conjunto de datos.

Por último, Visual Studio agrega una instancia de la nueva clase de conjunto de datos (**dsCategories1**) a la página.

Llegados a este punto, tiene configurado todo lo necesario para obtener información de la base de datos y almacenarla en un conjunto de datos.

## Agregar un control DataGrid para mostrar los datos

En este tutorial, agregará un único control, un control **DataGrid**, que puede mostrar todos los registros del conjunto de datos simultáneamente y que permite modificar registros.

La cuadrícula de datos debe enlazarse al conjunto de datos para mostrar los datos.

### Para agregar un control DataGrid al formulario

1. Si aún no lo ha hecho, cambie al Diseñador de Web Forms haciendo clic en la ficha, en la parte superior de la ventana actual.
2. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **DataGrid** al formulario.
3. Seleccione el control, presione F4 para mostrar la ventana Propiedades y haga clic en **Generador de propiedades** en la parte inferior de la ventana.

Aparece el cuadro de diálogo **Propiedades de DataGrid**.

4. En la ficha General, establezca la siguiente configuración:

Propiedad	Valor	Descripción
<b>DataSource</b>	dsCategories1	Enlaza la cuadrícula al conjunto de datos.
<b>DataMember</b>	Categories	Especifica que la cuadrícula debería mostrar los datos de la tabla de categorías en el conjunto de datos.

<b>Campo de clave de datos</b>	CategoryID	Especifica que la clave primaria del registro de categorías es la columna CategoryID. Esto le permitirá identificar el registro a actualizar en el conjunto de datos.
--------------------------------	------------	---

- Haga clic en **Aceptar** para cerrar el cuadro de diálogo **Propiedades de DataGrid**.
- Si desea cambiar la apariencia de la cuadrícula, configure las propiedades, por ejemplo, **Font** y **BackColor**.

**Sugerencia** Una forma sencilla de configurar la apariencia de la cuadrícula es haciendo clic en el botón **Formato automático** en la parte inferior de la ventana Propiedades y seleccionando un aspecto predefinido.

## Llenar un conjunto de datos y mostrar datos en el control DataGrid

Aunque la cuadrícula está enlazada con el conjunto de datos que creó, éste no se rellena automáticamente. Debe rellenarlo usted mismo llamando a un método de adaptador de datos. Para obtener información detallada sobre cómo llenar los conjuntos de datos, vea [Introducción a los conjuntos de datos](#).

Incluso después de llenar el conjunto de datos, el control **DataGrid** no muestra los datos automáticamente. Debe enlazar de manera explícita la cuadrícula a su origen de datos. Para obtener más información, vea [Introducción al enlace de datos en las páginas de formularios Web Forms](#).

### Para llenar el conjunto de datos y mostrar los datos en el control DataGrid

- Haga doble clic en la página para mostrar el archivo de clase de la página en el Editor de código.
- En el controlador de eventos **Page\_Load**, llame al método **Fill** del adaptador de datos, pasándole el conjunto de datos que desea llenar:

```
// Visual J#
sqlDataAdapter1.Fill(dsCategories1);
```

- Llame al método **DataBind** del control **DataGrid** para enlazar el control al conjunto de datos. Si no enlaza el control cada vez que la página realiza una acción de ida y vuelta al servidor, perdería los cambios del usuario en la cuadrícula. Por ello, debería enlazar la cuadrícula en estas ocasiones:

- La primera vez que se llama a la página.
- Cuando cambia el conjunto de datos.

Por ahora, vamos a enlazar la cuadrícula la primera vez que se llama a la página, probando la propiedad **IsPostBack** de la página. Agregue código en el controlador del evento **Page\_Load** después de llamar al método **Fill** del adaptador. El controlador completo puede tener el siguiente aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    sqlDataAdapter1.Fill(dsCategories1);
    if (! get_IsPostBack())
    {
        DataGrid1.DataBind();
    }
}
```

## Agregar la capacidad de modificar datos

Mientras se configura, el control **DataGrid** mostrará información de la tabla Categories. Pero también desea que los usuarios puedan modificar filas independientes en la cuadrícula. Para realizar esto, agregue un botón Edición a cada fila de la cuadrícula. El usuario hace clic en el botón y se vuelve a mostrar la cuadrícula en modo de edición, con cuadros de texto en los que los usuarios pueden modificar las columnas individuales. Cuando la fila se encuentra en modo de edición, el botón Edición se reemplaza por otros dos botones, un botón Actualizar y un botón Cancelar, como se muestra en la ilustración siguiente:

	CategoryID	CategoryName	Description
Actualizar Cancelar	1	Beverages	Soft drinks, coffees, teas,
Edición	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
Edición	3	Confections	Desserts, candies, and sweet breads
Edición	4	Dairy Products	Cheeses
Edición	5	Grains/Cereals	Breads, crackers, pasta, and cereal
Edición	6	Meat/Poultry	Prepared meats
Edición	7	Produce	Dried fruit and bean curd
Edición	8	Seafood	Seaweed and fish

Puede configurar las propiedades del control **DataGrid** para mostrar estos botones. Sin embargo, los botones no se enlazan automáticamente; en su lugar, cuando el usuario hace clic en un botón Edición, Actualizar o Cancelar, se provoca un evento al que responde el código. Es necesario agregar controladores de eventos que realicen lo siguiente:


- Botón Edición – establecer la fila actual en modo de edición.
- Botón Cancelar – devolver la fila a modo de presentación.

El controlador de evento para el botón Actualizar es más complejo porque lleva a cabo la actualización, que es el objetivo principal de este tutorial. Obtendrá más información al respecto en la siguiente sección.

### Para agregar la capacidad de modificar el control DataGrid

1. En la vista Diseño del Diseñador de Web Forms, seleccione el control **DataGrid**, presione F4 para mostrar la ventana Propiedades y haga clic en **Generador de propiedades** en la parte inferior de la ventana.

Se mostrará el cuadro de diálogo **Propiedades de DataGrid**.


2. Haga clic en la ficha **Columnas**.
3. Debajo de **Lista de columnas**, desplácese hacia abajo en la lista **Columnas disponibles** y abra el nodo **Columna Botón**.
4. Seleccione **Edición, Actualizar, Cancelar** y haga clic en el botón Agregar (  ) para agregar los botones al cuadro **Columnas seleccionadas**.
5. Haga clic en **Aceptar**.

El control **DataGrid** se vuelve a mostrar con un botón de vínculo Edición en la columna izquierda. (Al principio no verá los botones Actualizar y Cancelar).

Ahora que tiene un botón Edición, debe crear los controladores de eventos para establecer el modo de edición de las filas. Para controlar el modo de edición, establezca la propiedad **EditItemIndex** del control **DataGrid** en el índice (empezando por cero) de la fila que se va a modificar. Por ejemplo, para poner la tercera fila de la cuadrícula en modo de edición, debe establecer la propiedad en 2. Para devolver una fila al modo de presentación, debe establecer la propiedad en -1. Tras cambiar al modo de edición, debe enlazar de nuevo la cuadrícula para que muestre los datos de la fila.

Puede determinar en qué fila se encuentra el usuario conociendo el evento-objeto pasado al controlador. El objeto de evento de estos eventos contiene una propiedad **Item** que representa la toda la fila de **DataGrid** que se está actualizando. El objeto **Item** admite varias propiedades, incluyendo **Item.ItemIndex**, que devuelve el valor del índice de la fila en la que se está trabajando.

### Para configurar el modo de edición

1. En la vista Diseño, seleccione la cuadrícula y presione F4 para abrir la ventana Propiedades.
2. Haga clic en el botón Eventos (  ) de la parte superior de la ventana Propiedades.
3. Haga doble clic en **EditCommand** en la cuadrícula.

Se creará un controlador **DataGrid1\_EditCommand**.

4. Repita los pasos 2 y 3 para el evento **CancelCommand**.

Se creará un controlador **DataGrid1\_CancelCommand**.

5. En el controlador de evento **EditCommand**, agregue el código siguiente:

```
// Visual J#
DataGrid1.set_EditItemIndex(e.get_Item().get_ItemIndex());
DataGrid1.DataBind();
```

```
sqlConnection1.Close();
```

6. En el controlador de evento **CancelCommand**, agregue el código siguiente:

```
// Visual J#  
DataGrid1.set_EditItemIndex(-1);  
DataGrid1.DataBind();  
sqlConnection1.Close();
```

## Pruebas

Ahora tiene un control **DataGrid** que muestra datos desde un conjunto de datos (después de llenarlo con los datos de la base de datos) y ha configurado la cuadrícula con un botón Edición para cada fila. Aunque la página no esté aún terminada, es conveniente probarla en este momento para comprobar que la funcionalidad de edición trabaja correctamente.

### Para probar los pasos realizados hasta ahora

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página de formularios Web Forms y seleccione **Ver en el explorador**.

El proyecto se compila y la página de formularios Web Forms aparece en un panel de explorador en el diseñador. Si todo funciona correctamente, la cuadrícula se llenará con los datos y la primera columna contendrá vínculos con el texto "Edición".

2. Haga clic en el vínculo **Edición** de cualquier fila de la cuadrícula.

La fila se volverá a mostrar con los siguientes cambios:

- El vínculo Edición será reemplazado por vínculos Actualizar y Cancelar.
- Los datos se volverán a mostrar en controles **TextBox**.

3. Haga clic en **Cancelar**.

La fila se volverá a mostrar en la forma original.

La siguiente sección muestra los pasos a seguir para que el vínculo Actualizar escriba los cambios realizados en la cuadrícula en el conjunto de datos y en la base de datos.


## Actualizar el conjunto de datos y la base de datos

Hasta este punto, el control **DataGrid** ha hecho la mayor parte del trabajo para mostrar los datos. Sin embargo, como ocurre con otros controles de servidor ASP.NET, la cuadrícula no está capacitada para realizar actualizaciones automáticamente, es decir, enviar los cambios realizados por el usuario en la cuadrícula de vuelta al origen de datos. Para realizar las actualizaciones, será necesario escribir un código determinado.

La actualización se realiza realmente en dos fases. Primero, debe actualizar el conjunto de datos con los cambios realizados en la cuadrícula. Después debe escribir los cambios en el conjunto de datos en la base de datos. Para obtener más información, vea [Introducción a las actualizaciones de conjuntos de datos](#).

Cuando una fila de la cuadrícula se encuentra en modo de edición, la columna del extremo izquierdo contiene un vínculo Actualizar. Cuando los usuarios hacen clic en el vínculo, éste provoca un evento **UpdateCommand**. Escribirá todo el código de actualización en el controlador de este evento.

### Para crear un controlador de UpdateCommand

1. En la vista Diseño, seleccione la cuadrícula y presione F4 para abrir la ventana Propiedades.
2. Haga clic en el botón Eventos () de la parte superior de la ventana Propiedades.
3. Haga doble clic en **UpdateCommand** en la cuadrícula.

Se creará un controlador de evento **DataGrid1\_UpdateCommand**.

## Pasos para actualizar desde el control DataGrid

El esquema de todos estos pasos es el siguiente:

1. Determinar qué fila (mediante el índice) se actualizó en el control **DataGrid**. A continuación, obtener la clave de datos de la fila de la cuadrícula que se está actualizando para identificarla (por su identificador).
2. Obtener los valores modificados en la fila de la cuadrícula que actualizó el usuario.
3. Utilizar el valor de la clave de datos para encontrar la fila correspondiente en la tabla del conjunto de datos y, a continuación, escribir los cambios en dicha fila. En este punto, ha actualizado el conjunto de datos, pero todavía no ha actualizado la base de datos.
4. Enviar los cambios desde el conjunto de datos a la base de datos. Se ejecutará un comando SQL o un procedimiento almacenado que copiará los cambios en el conjunto de datos en la base de datos.
5. Actualizar el contenido del control **DataGrid**.
6. En las secciones siguientes se explican los detalles de cada uno de estos pasos. Si lo prefiere, puede omitir estas explicaciones e ir directamente al código.

#### Para actualizar desde el control **DataGrid**

1. Identifique qué fila del **DataGrid** se actualizó obteniendo la propiedad **ItemIndex** de la fila (objeto **Item**) pasada en el objeto de evento. Después utilice el valor del índice para obtener el valor correspondiente de la colección **DataKeys** de la cuadrícula:

```
// Visual J#  
String key = DataGrid1.get_DataKeys().get_Item(e.get_Item().get_ItemIndex()).ToString();
```

2. Obtenga los valores modificados de la fila del **DataGrid**. Para hacerlo:
  - a. Obtenga la celda apropiada (que empieza en cero) de la colección **Cells** del elemento pasado en el evento de objeto. Por ejemplo, la columna del extremo izquierdo de la cuadrícula es **Cells(0)**.
  - b. Obtenga la colección **Controls** de cada celda, que contiene todos los elementos que aparecen en la celda.
  - c. Obtenga el primer control (y sólo el primero) de la colección; en este caso, el control **TextBox**. Para obtener el control **TextBox**, declare una variable local de tipo **TextBox** y convierta el objeto de la colección **Controls** a este tipo.
  - d. Obtenga el valor del control **TextBox** (su propiedad **Text**).

En el ejemplo siguiente, se muestra cómo llevar a cabo estos pasos.

```
// Visual J#  
String categoryname;  
String categoryDescription;  
TextBox tb;  
tb = (TextBox)(((System.Web.UI.WebControls.TableCell)(e.get_Item().get_Cells().get_Item(2))).get_Controls().get_Item(0));  
categoryName = tb.get_Text();  
tb = (TextBox)(((System.Web.UI.WebControls.TableCell)(e.get_Item().get_Cells().get_Item(3))).get_Controls().get_Item(0));  
categoryDescription = tb.get_Text();
```

3. Encuentre la fila correspondiente en la tabla de datos. El conjunto de datos con tipo **dsCategories** contiene un método **FindBy** especial, en este caso, el método **FindByCategoryID**, que localiza una fila por su clave primaria y devuelve una referencia a la misma. Cree una variable para la fila de datos con tipo y llame al método:

```
// Visual J#  
dsCategories.CategoriesRow r;  
r = dsCategories1.get_Categories().FindByCategoryID(Integer.parseInt(key));
```

4. Actualice la fila cambiando los valores de la fila que encontró en el paso 3, como en el siguiente ejemplo:

```
// Visual J#  
r.set_CategoryName(categoryName);  
r.set_Description(categoryDescription);
```

5. Envíe los cambios desde el conjunto de datos a la base de datos llamando al método **Update** del adaptador de datos:

```
// Visual J#
```



```
sqlDataAdapter1.Update(dsCategories1);
DataGrid1.DataBind();
```

6. Cambie el modo de edición de la fila actual de la cuadrícula.

```
// Visual J#
DataGrid1.set_EditItemIndex(-1);
```

7. Enlace los datos al control **DataGrid**:

```
// Visual J#
DataGrid1.DataBind();
```

En el código siguiente se muestra el aspecto del controlador del evento **UpdateCommand**. Copie este código y péguelo en el archivo de clase de la página de formularios Web Forms.

**Sugerencia** Compruebe que sobrescribe el esqueleto del controlador de evento creado anteriormente. En caso contrario, tendrá dos métodos con nombres y firmas idénticos.

```
// Visual J#
private void DataGrid1_UpdateCommand (Object source, System.Web.UI.WebControls.DataGridCo
mmandEventArgs e)
{
    String categoryName, categoryDescription;

    // Gets the value of the key field of the row being updated
    String key = DataGrid1.get_DataKeys().get_Item(e.get_Item().get_ItemIndex()).ToString(
);

    // Gets get the value of the controls (textboxes) that the user
    // updated. The DataGrid columns are exposed as the Cells collection.
    // Each cell has a collection of controls. In this case, there is only one
    // control in each cell -- a TextBox control. To get its value,
    // you copy the TextBox to a local instance (which requires casting)
    // and extract its Text property.
    //
    // The first column -- Cells(0) -- contains the Update and Cancel buttons.
    TextBox tb;

    // Gets the value the TextBox control in the third column
    tb = (TextBox)(((System.Web.UI.WebControls.TableCell)(e.get_Item().get_Cells().get_Ite
m(2))).get_Controls().get_Item(0));
    categoryName = tb.get_Text();

    // Gets the value the TextBox control in the fourth column
    tb = (TextBox)(((System.Web.UI.WebControls.TableCell)(e.get_Item().get_Cells().get_Ite
m(3))).get_Controls().get_Item(0));
    categoryDescription = tb.get_Text();

    // Finds the row in the dataset table that matches the
    // one the user updated in the grid. This example uses a
    // special Find method defined for the typed dataset, which
    // returns a reference to the row.
    dsCategories.CategoriesRow r;
    r = dsCategories1.get_Categories().FindByCategoryID(Integer.parseInt(key));

    // Updates the dataset table.
    r.set_CategoryName(categoryName);
    r.set_Description(categoryDescription);
```

```
// Calls a SQL statement to update the database from the dataset
sqlDataAdapter1.Update(dsCategories1);

// Takes the DataGrid row out of editing mode
DataGrid1.set_EditItemIndex(-1);

// Refreshes the grid
DataGrid1.DataBind();
// Close the connection
sqlConnection1.Close();
}
```

## Pruebas

Ya ha completado todos los pasos. Debería probar la página para ver cómo funciona la cuadrícula y comprobar que actualiza los datos correctamente.

### Para probar la página

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página de formularios Web Forms y seleccione **Ver en el explorador**.

El proyecto se compila y la página de formularios Web Forms aparece en un panel de explorador en el diseñador.

2. Haga clic en el vínculo **Edición** de cualquier fila de la cuadrícula y modifique la fila mediante los cuadros de texto.
3. Haga clic en **Actualizar**.

Se volverá a mostrar la cuadrícula con los cambios. Si examina la base de datos, verá que también aparecen las modificaciones realizadas.

## Pasos siguientes

En este tutorial, no sólo se utilizó un control **DataGrid** en una página de formularios Web Forms para ver datos, sino también para modificarlos. Por cuestiones de claridad, el tutorial omite algunos detalles que debería tener en cuenta a la hora de diseñar una aplicación de producción. Algunas de las áreas en las que podría aumentar lo aprendido en este tutorial son:

### Mejoras en el control DataGrid

Este tutorial no obtiene todo el beneficio de las características más avanzadas del control DataGrid. Algunas de estas mejoras serían:

- Configurar columnas independientes en la cuadrícula seleccionando la presentación de determinadas columnas, modificando los encabezados de la columna, etc. Para obtener información detallada, vea [Agregar columnas enlazadas a un control DataGrid de servidor Web](#) y [Especificar el formato de un elemento de cuadrícula en un control DataGrid de servidor Web](#)
- Permite a los usuarios eliminar filas. Para obtener información detallada, vea [Permitir a los usuarios eliminar elementos en un control DataGrid de servidor Web](#). Además de especificar una botón Eliminar, es necesario agregar código para quitar los registros del conjunto de datos. Para obtener información detallada, vea [Eliminar registros de un conjunto de datos](#).
- Utilizar paginación. Hasta ahora, sólo se mostraban unos pocos registros. Sin embargo, normalmente se deben mostrar muchos más y, en ese caso, conviene que la cuadrícula muestre la información por páginas. Para obtener información detallada, vea [Comportamiento de la paginación en los controles DataGrid de servidor Web](#).

## Almacenar en caché el conjunto de datos

En el tutorial, cada vez que la página realiza una acción de ida y vuelta al servidor, se vuelve a crear una instancia del conjunto de datos y se vuelve a llenar con los datos de la base de datos. Esta estrategia, sencilla de implementar, puede provocar tráfico innecesario en el servidor de la base de datos.

Una alternativa consiste en almacenar el conjunto de datos después de llenarlo y cada vez que se modifica. Después, en lugar de volver a leer la información de la base de datos, puede restaurar el conjunto de datos en cada acción de ida y vuelta al servidor. Existen distintas opciones para almacenar el conjunto de datos, por ejemplo, incluirlo en el estado de sesión (en el servidor) o en

el estado de vista (en el cliente, en la página). Para obtener información detallada sobre cada estrategia, vea [Recomendaciones de administración de estado](#).

Por ejemplo, si desea almacenar el conjunto de datos en el estado de sesión, debería incluir código como el siguiente en el controlador del evento **Page\_Load**:

```
// Visual J#
if (this.get_Session().get_Item("mydataset")== null)
{
    sqlDataAdapter1.Fill(dsCategories1);
    this.get_Session().set_Item("mydataset", dsCategories1);
}
else
{
    dsCategories1 = (dsCategories) this.get_Session().get_Item("mydataset");
}
if (! get_IsPostBack())
{
    DataGrid1.DataBind();
}
```

Observe que si almacena el conjunto de datos en el estado de sesión o de aplicación, debe convertirlo al tipo de conjunto de datos apropiado cuando lo recupera.

Compruebe que actualiza la copia del conjunto de datos en el estado de sesión cada vez que se modifica. Por ejemplo, podría agregar la siguiente línea al controlador del evento **UpdateCommand** inmediatamente después de actualizar la base de datos:

```
// Visual J#
sqlDataAdapter1.Update(dsCategories1);
this.get_Session().set_Item("mydataset", dsCategories1);
```

## Utilizar acceso directo a la base de datos en lugar de un conjunto de datos

En este tutorial, se creó un conjunto de datos que se enlazó con una cuadrícula. Para actualizar la base de datos, primero actualizó el conjunto de datos y después utilizó el método **Update** del adaptador de datos para enviar los cambios a la base de datos.

Con un conjunto de datos es fácil realizar actualizaciones, pero supone un alto costo de memoria y un rendimiento bajo. Como alternativa, puede utilizar comandos de datos (objetos [Clase OleDbCommand](#) o [Clase SqlCommand](#)) para ejecutar instrucciones SQL o procedimientos almacenados directamente. Por ejemplo, puede configurar un comando de datos con una instrucción Update de SQL parametrizada para actualizar la tabla Categories en la base de datos. Después de obtener las modificaciones realizadas en la cuadrícula, establece los parámetros para la instrucción Update y la ejecuta.

Para obtener más información, vea [Introducción a los objetos DataCommand en Visual Studio](#)

## Validar datos

En una aplicación de producción, probablemente incluiría comprobaciones de validación para asegurarse de que todas las modificaciones realizadas por el usuario son correctas. Para obtener más información, vea [Validación de datos en los conjuntos de datos](#).

## Control de concurrencia

En este tutorial, se enviaban las actualizaciones a la base de datos sin tener en cuenta si el registro actualizado también había sido modificado por otro usuario. Sin embargo, en las aplicaciones de producción, se incluyen normalmente comprobaciones de errores para mantener la integridad de la base de datos. Para obtener información detallada, vea [Control de concurrencia en ADO.NET](#).

## Vea también

[Tutorial: mostrar datos en una página de formularios Web Forms con Visual J# |](#)

[Tutorial: crear acceso a datos de sólo lectura en una página de formularios Web Forms con Visual J# |](#)

[Tutorial: crear una aplicación distribuida con Visual J# |](#)

[Tutorial: actualizar datos mediante una consulta de actualización de bases de datos en los formularios Web Forms con Visual J# |](#)

[Tutorial: validar los datos proporcionados por el usuario en una página de formularios Web Forms con Visual J# |](#)



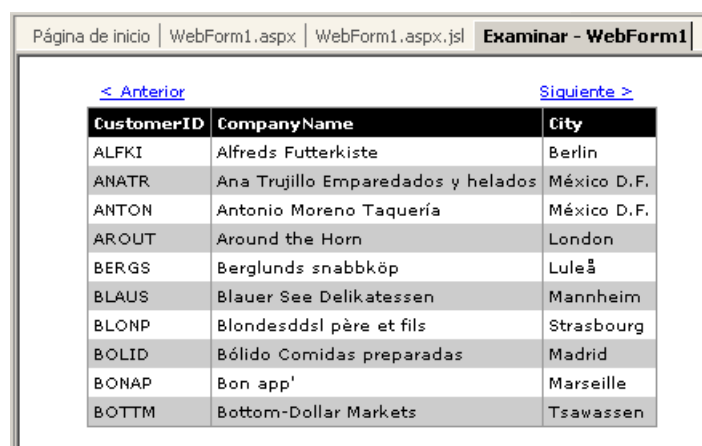
# Tutorial: Crear acceso a datos paginados mediante una página de formularios Web Forms con Visual J#

Un escenario frecuente en las aplicaciones Web es mostrar una lista, por ejemplo, una lista de resultados de la búsqueda de productos de un catálogo. Salvo que la lista sea corta, ésta se suele mostrar en páginas, proporcionando al usuario una manera de desplazarse entre ellas. Este tutorial muestra una forma de crear resultados paginados utilizando un control **DataGrid** de servidor Web.

Puede crear resultados paginados de diferentes formas:

- Creando y llenando un conjunto de datos para cada página. Esta estrategia es sencilla, permitiendo aprovechar las características de paginación integradas en el control **DataGrid**. Sin embargo, no es muy eficiente, ya que obtiene la tabla completa de la base de datos cada vez que el usuario se desplaza a otra página. Para obtener información detallada, vea [Especificación del comportamiento de la paginación en un control DataGrid de servidor Web](#).
- Creando un conjunto de datos y almacenándolo en memoria caché. Puede crear un conjunto de datos y después almacenarlo (por ejemplo, en la vista de sesión). Esto evita la sobrecarga de obtener la tabla de la base de datos cada vez, pero tiene el inconveniente de almacenar la tabla completa en la memoria de servidor. Para tablas pequeñas, ésta forma resulta eficiente.
- Obteniendo una página con los datos que se necesiten de la base de datos. Este método es eficaz por dos razones: transfiere sólo la cantidad necesaria de datos cada vez y no hay que almacenar un conjunto de datos en la memoria de servidor. Sin embargo, es la forma más compleja de crear.

En este tutorial se explicará el último método. Cuando termine, la página tendrá el siguiente aspecto:



<a href="#">&lt; Anterior</a>			<a href="#">Siguiente &gt;</a>		
CustomerID	CompanyName	City			
ALFKI	Alfreds Futterkiste	Berlin			
ANATR	Ana Trujillo Emparedados y helados	México D.F.			
ANTON	Antonio Moreno Taquería	México D.F.			
AROUT	Around the Horn	London			
BERGS	Berglunds snabbköp	Luleå			
BLAUS	Blauer See Delikatessen	Mannheim			
BLONP	Blondesddsl père et fils	Strasbourg			
BOLID	Bólide Comidas preparadas	Madrid			
BONAP	Bon app'	Marseille			
BOTTM	Bottom-Dollar Markets	Tsawassen			

Para completar este tutorial, necesitará:

- Acceso a un servidor con la base de datos de ejemplo Northwind de SQL Server.
- Los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

El tutorial está dividido en varias partes más pequeñas:

- Crear un proyecto de aplicación Web y una página de formularios Web Forms.
- Agregar los componentes de datos necesarios para obtener los datos de la base de datos.
- Agregar un control **DataGrid** al formulario.
- Agregar código para obtener los datos correctos cuando los usuarios se desplazan de una página a otra.

Como este tutorial se centra en conceptos avanzados más complejos de las páginas de formularios Web Forms, debe estar familiarizado con los siguientes conceptos:

- Comandos de datos. Para obtener información detallada, vea [Introducción a los objetos DataCommand en Visual Studio](#).
- Configurar parámetros de comandos. Para obtener más información, vea [Establecer y obtener parámetros de comandos de datos](#).
- Utilizar un lector de datos como un cursor de sólo lectura y de sólo avance. Para obtener información detallada, vea [Ejecutar un comando de datos que devuelve un conjunto de resultados](#).
- Guardar la información de la página en el estado de vista entre las acciones de ida y vuelta al servidor. Para obtener información detallada, vea [Guardar los valores de una página de formularios Web Forms mediante el estado de vista](#).

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo `http://`, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada `WebForm1.aspx` en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Agregar los componentes de datos

En este tutorial, deberá ejecutar dos instrucciones SQL diferentes para obtener una página de datos. Ambas instrucciones obtienen 10 filas cada vez basándose en un identificador inicial; difieren sólo en la cláusula WHERE.

- La instrucción para obtener la siguiente página de datos, obtiene 10 filas de la tabla empezando por la fila posterior (es decir, que es mayor que) la última de la página actual.
- La instrucción para obtener la página anterior obtiene 10 registros empezando por (es decir, que es igual a) el primero de la página anterior.

**Nota** Como alternativa a utilizar dos instrucciones, puede crear una sola instrucción y cambiar la cláusula WHERE dinámicamente. En este tutorial, utilizará dos instrucciones independientes para reducir el código.

En este tutorial, trabajará con la tabla Customers de la base de datos Northwind de SQL Server. Puede utilizar las técnicas explicadas en el tutorial para cualquier tabla, teniendo en cuenta las siguientes condiciones:

- La tabla a la cual está teniendo acceso tiene un identificador único.
- La tabla está ordenada por el identificador.
- El lenguaje SQL utilizado admite la palabra clave TOP en la instrucción Select. La palabra clave TOP le permite especificar el número de filas a devolver. En este caso, utilizará la palabra clave TOP para especificar el número de filas por página.

## Crear y configurar la conexión

Para comenzar, agregue una conexión que señale a la base de datos de la que desea leer.

### Para crear la conexión de datos

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre un objeto **SqlConnection** a la página.

**Nota** Como está utilizando SQL Server, se utiliza el objeto **SqlConnection**, que está optimizado para trabajar con SQL Server 7.0 o posterior. Si está utilizando una base de datos diferente, utilizará el objeto **OleDbConnection**, que proporciona acceso ADO.NET a cualquier origen de datos compatible con OLE DB.

2. Seleccione la conexión y, en la ventana **Propiedades**, haga clic en el botón en el cuadro de propiedades **ConnectionString**. La lista muestra todas las conexiones existentes y la opción **<Nueva conexión...>**.
3. Si aún no tiene una conexión a la base de datos Northwind de SQL Server, seleccione **<Nueva conexión...>**. Se mostrará el cuadro de diálogo **Propiedades de vínculo de datos**.
4. Cree una conexión que señale a SQL Server y a la base de datos Northwind y, a continuación, haga clic en **Aceptar**.

**Nota** Necesita permisos de lectura/escritura en el servidor de SQL Server que esté utilizando. Se recomienda

especificar la seguridad integrada de Windows cuando se crea la conexión. Alternativamente, puede especificar un nombre de usuario y una contraseña y guardar dicha información con la conexión, pero eso puede afectar a la seguridad. Para obtener más información, vea [Acceso a SQL Server desde una aplicación Web](#).

La información de conexión especificada se guarda como cadena de conexión de la conexión.

Ahora es necesario agregar comandos. Necesitará dos en total, uno para cada instrucción SQL.

#### Para crear los comandos de datos para obtener páginas de datos

1. Desde la ficha **Datos** del Cuadro de herramientas, arrastre dos objetos **SqlCommand** a la página.
2. Llame a uno **cmdNext** y al otro **cmdPrevious**.
3. Seleccione el comando **cmdNext** y, en la ventana Propiedades, haga clic en el botón de la propiedad **Conexiones** y elija la conexión realizada anteriormente. (Deberá abrir el nodo **Existentes**).
4. En la propiedad **CommandText**, haga clic en el botón de puntos suspensivos para abrir el cuadro de diálogo **Generador de consultas**.
5. Utilice el cuadro de diálogo **Generador de consultas** para construir la siguiente instrucción SQL:

```
SELECT TOP 10 CustomerID, CompanyName, City
FROM Customers
WHERE (CustomerID > @customerid)
ORDER BY CustomerID
```

6. Cuando termine, cierre el cuadro de diálogo **Generador de consultas**.

La instrucción SQL rellena la propiedad **CommandText**.

7. Repita los pasos que van del 3 al 5 para el objeto de comando **cmdPrevious**, creando esta consulta:

```
SELECT TOP 10 CustomerID, CompanyName, City
FROM Customers
WHERE (CustomerID >= @customerid)
ORDER BY CustomerID
```

Ha terminado de agregar componentes de datos a la página.

#### Agregar el control DataGrid

Para mostrar la paginación, necesitará un control **DataGrid**. Para el desplazamiento, creará los botones Anterior y Siguiente.

#### Para agregar y configurar el control DataGrid

1. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **DataGrid** a la página.
2. En la parte inferior de la ventana Propiedades, seleccione el vínculo **Formato automático** y elija un formato predefinido para la cuadrícula.
3. En la ventana Propiedades, haga clic en el vínculo **Generador de propiedades**.

Aparece el cuadro de diálogo **Propiedades de DataGrid1**.

4. Haga clic en la ficha **Paginación** y establezca lo siguiente:

Propiedad	Valor
Permitir paginación	Checked
Permitir paginación personalizada	Checked
Tamaño de página	10
Mostrar botones de exploración	Unchecked

5. Haga clic en **Aceptar** para cerrar el cuadro de diálogo.

**Nota** Si ya ha trabajado antes con un control **DataGrid** de datos enlazados, es posible que piense configurar la propiedad **DataSource** del control. Sin embargo, en este ejemplo establece el origen de datos en el código. De manera similar, tampoco establecerá las columnas.

#### Para agregar botones de desplazamiento

1. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre dos controles **LinkButton** a la página.
2. Llame a uno **btnPrevious** y al otro **btnNext**.
3. Establezca el texto de los botones. Por ejemplo, el texto del botón Siguiente podría ser `Siguiente >`.

## Agregar código para buscar y mostrar una página de datos

Ahora se puede agregar código para realizar la paginación. La lógica se basa en que, mientras los usuarios se desplazan por las páginas, se ejecutará una instrucción SQL (utilizando los objetos de comandos de datos creados) y se devolverá un lector de datos (objeto `SqlDataReader`) con 10 filas. El control **DataGrid** se enlaza al lector de datos y las filas se muestran automáticamente en la cuadrícula.

## Página siguiente y página anterior

La única complicación de la página radica en implementar el desplazamiento, y el problema es el siguiente: ¿cómo determinar los 10 registros que se deben obtener de la base de datos?

La lógica para la página Siguiente debe obtener 10 filas empezando por la fila inmediatamente posterior a la última fila de la cuadrícula. Para ello obtendrá el identificador de la última fila de la cuadrícula y lo utilizará como parámetro en el comando `cmdNext` creado anteriormente.

La lógica para desplazarse a la página anterior es más compleja. Es necesario obtener 10 filas empezando por (e incluyendo) el primer identificador de la página anterior a la que se está mostrando en ese momento en la cuadrícula. Para ello, el identificador de la página anterior deberá estar disponible, y si el usuario continúa desplazándose hacia atrás, también el de las páginas anteriores.

Una solución es crear una estructura que almacene el identificador del primer registro de cada página mientras se esté mostrando esa página. De esa forma, cuando se vuelve a la página, se puede obtener su primer identificador y utilizarlo como parámetro para la consulta que llena la página.

Como se trata de una página de formularios Web Forms, no es posible almacenar la información de la página en una variable; debe almacenarla en algún sitio donde se conserve entre las acciones de ida y vuelta al servidor. En este tutorial, se almacena la información en el estado de vista, que la coloca en la propia página (en un campo oculto). Deberá almacenar dos tipos de elementos:

- El número de la página actual. Lo utilizará como un valor de índice, y también le informará cuándo el usuario se ha desplazado a la primera página.
- En cada página, el identificador de la primera fila de la cuadrícula. Creará un nuevo elemento en el estado de vista para cada página. El nombre de este elemento es el número de página, y su valor es el identificador.

Por ejemplo, si se encuentra en la tercera página de la cuadrícula, se habrán agregado cuatro elementos al estado de vista, algo parecido a esto:

```
CurrentPage=3
0=ALFKI
1=BSBEV
2=FAMIA
```

Las secciones siguientes le muestran el código necesario para implementar la lógica de paginación.

## Mostrar datos en la cuadrícula

Para llenar la cuadrícula no importa cómo se desplaza el usuario. Por lo tanto, puede crear un método genérico que ejecute los comandos `cmdNext` o `cmdPrevious`, dependiendo de cuál de ellos se haya pasado al método.

## Para mostrar los datos en la cuadrícula

1. Haga clic con el botón secundario en la superficie del Diseñador de WebForm1 y haga clic en **Ver código** para abrir `WebForm1.aspx.jsl`.
2. Crear una variable privada denominada `CurrentPage` de tipo `int`. El valor de esta variable se establecerá en otros métodos posteriores. En el ejemplo siguiente se muestra cómo declarar la variable `CurrentPage`.

```
// Visual J#
public class WebForm6 extends System.Web.UI.Page
{
```



```
private int CurrentPage;
// more declarations here
```

3. Agregue el siguiente código para crear un método denominado `FillGrid` que acepte un comando de datos (`SqlCommand`) como parámetro. Este método debe seguir al método `OnInit`.

```
// Visual J#
private void FillGrid(System.Data.SqlClient.SqlCommand currentSqlCommand)
{
    System.Data.SqlClient.SqlDataReader dr;
    sqlConnection1.Open();
    dr =currentSqlCommand.ExecuteReader();
    DataGrid1.set_DataSource(dr);
    DataGrid1.DataBind();
    dr.Close();
    sqlConnection1.Close();
    this.get_ViewState().set_Item("CurrentPage", (System.Int32)CurrentPage);
    this.get_ViewState().set_Item(((System.Int32)CurrentPage).ToString(),
        ((System.Web.UI.WebControls.TableCell) DataGrid1.getItems().getItem(0).get_Cells(
).getItem(0)).get_Text());
    // Determine how many rows were filled into the grid. If it is less
    // than the number of rows per page, there are no more rows in the
    // table, and the Next button should be disabled.
    if (DataGrid1.getItems().get_Count() < DataGrid1.get_PageSize())
    {
        btnNext.set_Enabled(false);
    }
}
```

El método `FillGrid` lleva a cabo las siguientes tareas:

- Cree una instancia de un lector de datos.
- Abre la conexión y el método `ExecuteReader` del comando de datos para ejecutar la instrucción SQL y obtener las filas.
- Establece la propiedad **DataSource** del control **DataGrid** en el lector de datos y, después, enlaza la cuadrícula.
- Cierra el lector de datos y la conexión.
- Guarda dos valores en el estado de vista:
  - El valor de la variable `CurrentPage`.
  - El valor de la columna de identificador de la primera fila de la cuadrícula. El valor se almacena en un elemento del estado de vista cuyo nombre es el número de página.
- Determina si se encuentra al final de la tabla. Para que sea más sencillo, en este tutorial, el código sólo prueba si la consulta devuelve menos filas que el tamaño de la página (10). Si se alcanza el final de la tabla, desactiva el botón Siguiente para que los usuarios no intenten continuar.

**Nota** Para consultar otras formas que le permitan determinar si se encuentra al final de la tabla, vea "Pasos siguientes" en este tutorial.

## Inicializar la página

La primera vez que se ejecuta la página, debe mostrar la primera página de datos. Ahora que ya dispone del método genérico para llenar la cuadrícula, todo lo que necesita es establecer un parámetro y llamar al método. Desea ejecutar el comando `cmdNext`. Establezca su parámetro `@customerid` en una cadena vacía, lo que significa que la consulta devolverá los primeros 10 registros mayores que "", es decir, los primeros 10 registros de la tabla. También debe establecer el número de página (`CurrentPage`) en cero para indicar que se encuentra en la primera página.

El código para inicializar la página está en `WebForm1.aspx.jsl` y tiene el siguiente aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
```

```
// Put user code to initialize the page here
if (!get_IsPostBack())
{
    ((System.Data.SqlClient.SqlParameter)(((System.Data.SqlClient.SqlParameterCollection)cmdNext.get_Parameters()).get_Item("@customerid"))).set_Value(System.String.Empty);
    CurrentPage = 0;
    FillGrid(cmdNext);
}
}
```

## Desplazarse a la página siguiente

Cuando los usuarios se desplazan a la página siguiente, es necesario ejecutar de nuevo el comando `cmdNext` utilizando el método genérico. En esta ocasión, el parámetro para el comando es el valor de la columna de identificador de la última fila de la cuadrícula.

También debe aumentar el número de páginas para que el método `FillGrid` pueda utilizar el número para crear otro elemento del estado de vista para el Id. Para aumentar el número de páginas, debe quitar el antiguo del estado de vista. Cuando se realiza esto, debe convertirlo a un entero, ya que todos los elementos del estado de vista están almacenados como objetos.

El código para el botón Siguiente está en `WebForm1.aspx.jsl` y tiene el siguiente aspecto:

```
// Visual J#
private void btnNext_Click (Object sender, System.EventArgs e)
{
    // Get the page number of the page most recently displayed
    CurrentPage = (int)((System.Int32)(this.get_ViewState().get_Item("CurrentPage")));
    CurrentPage++;
    // Gets the id on current page
    String lastid = ((System.Web.UI.WebControls.TableCell) DataGrid1.get_Items().get_Item(9).get_Cells().get_Item(0)).get_Text();
    ((System.Data.SqlClient.SqlParameter)(((System.Data.SqlClient.SqlParameterCollection)cmdNext.get_Parameters()).get_Item("@customerid"))).set_Value(lastid);
    FillGrid(cmdNext);
}
```

## Desplazarse a la página anterior

Por último, necesita agregar código para desplazarse hacia atrás. Aquí es cuando se utiliza la información almacenada en el estado de vista. Primero, obtiene el número de la página actual y después lo disminuye. Después busca en el estado de vista, utilizando el número de página como nombre, para obtener el identificador de la primera fila de la página. El identificador se utiliza como parámetro en el comando `cmdPrevious`, y se pasa al método genérico `FillGrid`.

Para evitar el intento de desplazarse más allá del principio de la tabla, debe comprobar que el contador de páginas no es menor que cero. Además, debería habilitar el botón Siguiente en el caso de que estuviera deshabilitado en el desplazamiento anterior.

El código para el botón Anterior está en `WebForm1.aspx.jsl` y tiene el siguiente aspecto:

```
// Visual J#
private void btnPrevious_Click (Object sender, System.EventArgs e)
{
    btnNext.set_Enabled( true);
    CurrentPage =(int)(System.Int32) (this.get_ViewState().get_Item("CurrentPage"));
    CurrentPage--;
    if (CurrentPage >= 0)
    {
        String firstid;
        firstid = (String) this.get_ViewState().get_Item(((System.Int32)CurrentPage).ToString());
    };
    ((System.Data.SqlClient.SqlParameter)(((System.Data.SqlClient.SqlParameterCollection)cmdPrevious.get_Parameters()).get_Item("@customerid"))).set_Value(firstid);
    FillGrid(cmdPrevious);
}
}
```

Tras crear los botones Anterior y Siguiente, debe asociar los eventos Click de los mismos con sus respectivos métodos de controlador.

### Para asociar los eventos Click con los métodos de controlador

1. Resalte el botón Anterior y haga clic en el botón **Eventos** de la ventana **Propiedades**. En **Acción**, defina el evento **Click** como **btnPrevious\_Click**.
2. Haga lo mismo para el botón Siguiente, pero defínalo como **btnNext\_Click**.

### Pruebas

Después de agregar el código, pruebe el acceso a datos en la página de formularios Web Forms.

### Para probar la página de formularios Web Forms

1. Guarde la página.
2. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la página y elija **Ver en el explorador**.

Compruebe que en la cuadrícula se muestra una lista de 10 clientes.

3. Haga clic en los botones Siguiente y Anterior varias veces para comprobar que el desplazamiento es correcto. Puede realizar las siguientes pruebas:
  - En la primera página, haga clic en Anterior; no debería ocurrir nada.
  - Haga clic en Siguiente hasta que alcance el final de la tabla para comprobar que se deshabilita el botón Siguiente.

### Pasos siguientes

En este tutorial se ha explicado un método para implementar la paginación. El escenario mostrado era sencillo pero completo. En las aplicaciones reales, pueden entrar en juego muchos factores adicionales y complicar la paginación. Algunas cuestiones a considerar serían las siguientes:

- Para determinar cuándo se ha alcanzado el final del archivo, el tutorial utiliza un algoritmo sencillo pero no del todo confiable. Compara el número de filas devueltas con el número de filas de la página. En una aplicación real, podría ser necesario agregar un algoritmo más sofisticado. Una posibilidad es realizar lo siguiente:
  1. Agregar otro objeto de comando a la página que incluya esta instrucción SQL:

```
Select Count(CustomerID) From Customers
```
  2. La primera vez que se ejecuta la página, ejecutar el comando utilizando el método **ExecuteScalar** del comando para obtener el número total de filas de la tabla. Después, calcular el número total de páginas necesarias para la tabla y almacenarlo en una variable global y en el estado de vista. En los envíos posteriores, obtener el valor del estado de vista.
  3. En el método `FillGrid`, comprobar si la página actual ha alcanzado el número total de páginas. En ese caso, deshabilitar el botón Siguiente.
- Este tutorial asume que los datos no son volátiles. Si otros usuarios están agregando filas a la tabla cambian los límites de la página, por lo que no es una solución viable almacenar el primer identificador de la página.
- Puede crear un procedimiento almacenado para realizar el trabajo de las instrucciones SQL.

### Vea también

[Establecer y obtener parámetros de comandos de datos](#) | [Ejecutar un comando de datos que devuelve un conjunto de resultados](#) | [Tutoriales sobre los formularios Web Forms](#)

# Tutorial: crear un control Web de usuario con Visual J#

Los controles de usuario Web se pueden definir según las necesidades de las aplicaciones y mediante las mismas técnicas de programación utilizadas para escribir las páginas de formularios Web Forms. Estos controles se pueden crear cuando se necesiten partes reutilizables de interfaz de usuario que se usarán en toda la aplicación. Un control de usuario puede contener código HTML, controles de servidor y lógica para la administración de eventos. Al igual que un control compilado, puede exponer propiedades que pueda establecer la página de formularios Web Forms que lo aloja.

**Nota** Los controles de usuario Web no deben confundirse con los controles Web personalizados. Para obtener más información, vea [Recomendaciones sobre los controles Web de usuario y los controles Web personalizados](#)

Los pasos para crear un control de usuario son muy similares a los realizados para crear una página de formularios Web Forms. La interfaz de usuario se diseña de forma visual organizando los controles de servidor ASP.NET, el código HTML y el texto estático en la superficie de diseño, enlazando los datos y escribiendo código para controlar los eventos provocados por los controles. Como en las páginas de formularios Web Forms, los controles de usuario pueden gestionar eventos de procesamiento de página como **Page\_Load**.

Este tutorial crea un control de menú de selección que pasa el elemento seleccionado a la página en la cadena de consulta de la dirección URL. Este estilo de desplazamiento resulta útil cuando la aplicación usa una sola página para mostrar una clase de información determinada, pero además se quiere proporcionar una vista filtrada de los datos. Por ejemplo, si se tiene una página Sales que muestra datos que se pueden filtrar por región, un usuario podría desplazarse directamente a los datos que desee ver haciendo clic en un botón que aplique un filtro sobre la dirección URL como un elemento de la cadena de consulta, como en este ejemplo que limita la información que se muestra a las ventas en el noroeste:

```
http://sales/salesreport.aspx?region=northeast
```

En este tutorial, creará un control que muestra categorías de información como un menú en una página de formularios Web Forms; a continuación, obtiene las categorías y rellena el menú de forma dinámica a partir del origen de datos de la página, en este caso una matriz de cadenas. Cuando se hace clic en un elemento del menú, el número de índice en la matriz de dicho elemento se agrega como parámetro a una dirección URL que se pasa a la página que aloja el control, ésta toma el parámetro y muestra la información que está asociada con el número de índice en un control Label; en este caso, la información es simplemente el nombre de la categoría de la misma matriz de cadenas.

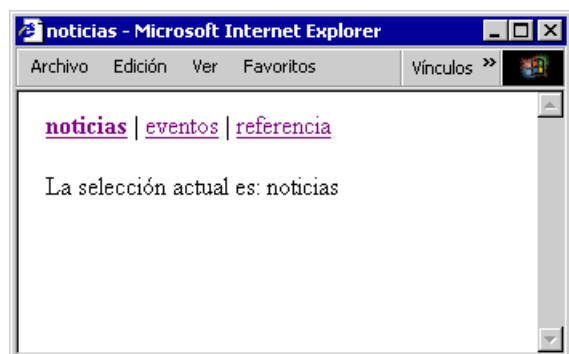
Para crear el control de usuario seguirá estos pasos:

- Abrir un nuevo control de usuario en el diseñador.
- Establecer una expresión de enlace de datos para el control de usuario.
- Exponer el origen de datos y la selección como propiedades a una página de formularios Web Forms.
- Inicializar el control de usuario.

Asimismo, hará lo siguiente para usar el control en una página de formularios Web Forms:

- Agregar el control de usuario a la página.
- Crear un origen de datos para el control de usuario.
- Establecer las propiedades del control de usuario.
- Proporcionar una presentación filtrada en la página de formularios Web Forms, basada en la selección del control de usuario.
- Ejecutar la página de formularios Web Forms para probar el control.

Al finalizar, la página podría tener el siguiente aspecto:



## Crear el control de usuario

El primer paso consiste en iniciar un nuevo proyecto y agregarle un control de usuario Web.

### Para crear el control de usuario Web

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET** y haga clic en **Aceptar**.  
Se creará un nuevo proyecto con el nombre y en la ubicación predeterminados.
3. Cierre el archivo .aspx recién creado. No lo necesitará en este tutorial.
4. En el menú **Proyecto**, haga clic en **Agregar control de usuario Web**.  
Aparecerá el cuadro de diálogo **Agregar nuevo elemento**, con **Control de usuario Web** seleccionado.
5. En el cuadro **Nombre**, escriba **menu.ascx** y haga clic en **Abrir**. El control de usuario se abrirá en el diseñador.

Para mostrar los datos en el control de usuario, agregue un control **DataList** a la superficie de diseño. El control **DataList** toma todos los elementos de un origen de datos, les da formato y los organiza según las propiedades especificadas, y muestra la lista de elementos. Debido a que la lista se genera de forma dinámica basándose en el origen de datos, este control se puede volver a utilizar fácilmente.

### Para agregar el control DataList y establecer sus propiedades

1. En la ficha **Web Forms** del Cuadro de herramientas, agregue un control **DataList** a la página menu.ascx.  
Este control rellenará el menú mostrando una lista de opciones disponibles, que obtendrá de una matriz que se crea en la página de formularios Web Forms. Para hacer que los elementos aparezcan en la página en el formato de menú "Elemento1 | Elemento2 | Elemento3", se deben establecer correctamente las propiedades de presentación del control **DataList**.
2. Asegúrese de que el control **DataList** está seleccionado y haga clic en **Generador de propiedades** en la parte inferior de la ventana Propiedades para abrir el cuadro de diálogo **Propiedades de DataList1**.
3. En la página **General** del grupo **Repetir el diseño**, establezca **Dirección** en **Horizontal**.
4. En la página **Formato**, en la lista **Objetos**, expanda el nodo **Elementos** y seleccione **Elementos seleccionados**. En el grupo **Apariencia**, seleccione **Negrita** y, a continuación, haga clic en **Aceptar**.  
Esta configuración hará que el elemento de menú seleccionado aparezca en negrita.
5. Haga clic con el botón secundario del *mouse* (ratón) en el control **DataList** para mostrar el menú contextual; a continuación elija **Editar plantilla** y haga clic en **Plantilla del separador**.
6. Sitúe el punto de inserción en la fila en blanco de la plantilla del **Separador** y escriba el carácter "|".  
Este carácter se usará para separar los elementos de menú cuando aparezcan en la lista.

Cada elemento de datos que aparece en el menú **DataList** tendrá un valor de índice de matriz con un enlace inferior de 0. Debe utilizar este número para solicitar la información asociada con ese elemento de menú, agregándolo como un parámetro a la dirección URL que se pasa a la página host. Para generar la dirección URL, dé a los elementos de datos el formato de hipervínculos mediante el control **HyperLink** y enlace ese control con el origen de datos. A continuación, puede diseñar una expresión de enlace que genere dinámicamente la cadena de petición adecuada para cada hipervínculo.

### Para agregar un control HyperLink y establecer la expresión de enlace

1. Haga clic con el botón secundario del *mouse* en el control **DataList** y, a continuación, elija **Editar plantilla** y haga clic en **Plantillas de elementos**. En el control **DataList** aparecerán varios campos de plantilla más.  
El usuario tiene que poder hacer clic en un elemento de menú para recuperar los datos relacionados con dicho elemento de menú, por tanto, cada elemento tiene que ser un hipervínculo.
2. En la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **HyperLink** a la fila en blanco del control **DataList** situada debajo del encabezado **ItemTemplate**.
3. Asegúrese de que el control **HyperLink** está seleccionado y, a continuación, en la ventana Propiedades, seleccione **(DataBindings)** y haga clic en el botón de puntos suspensivos del campo del valor.

Aparecerá el cuadro de diálogo **DataBindings**. La propiedad **Text** estará seleccionada de forma predeterminada en la lista **Propiedades enlazables**.

4. En el grupo **Enlace para Text**, bajo **Enlace simple**, expanda el nodo **Container** y seleccione **DataItem**.

Esto enlazará la propiedad **Text** de cada hipervínculo con el elemento de datos del contenedor, que en este caso es el control **DataList**. El control **DataList** recibirá sus elementos de datos de la matriz creada en la página de formulario Web Forms.

5. En la lista **Propiedades enlazables**, seleccione la propiedad **NavigateUrl**.

**Nota** Cuando haga clic en la propiedad **NavigateUrl**, el icono situado junto a la propiedad **Text** cambiará para mostrar que está enlazado a datos.

6. En el grupo **Enlace para NavigateUrl**, bajo **Enlace simple**, seleccione **ItemIndex**.

Éste es el número de índice del elemento de datos en la matriz. Para incorporar correctamente este número a la dirección URL de forma que ésta recupere el elemento de datos con ese número de índice, deberá asignarle el formato de un parámetro. Denomine al parámetro "category".

7. En el cuadro **Formato**, escriba esta cadena (observe las llaves antes y después del 0):

```
?category={0}
```

Ahora deberá completar la dirección URL de forma que señale a la página de formulario Web Forms que aloja el control.

8. Haga clic en **Expresión de enlace personalizada** y agregue este código al principio de la expresión de enlace, que está casi terminada:

```
get_Request().get_Path() +
```

La expresión de enlace de datos completa debería ser la siguiente:

```
get_Request().get_Path() + DataBinder.Eval(Container, "ItemIndex", "?category={0}")
```

9. Haga clic en **Aceptar** para cerrar el Cuadro de diálogo DataBindings.

La expresión de enlace de datos que ha escrito contiene la sintaxis para generar una dirección URL en tiempo de ejecución, basándose en información del vínculo asociado con el número de la matriz.

Cuando el control de usuario se ejecute, esta expresión se evaluará del modo siguiente:

- Primero, se reemplaza **get\_Request().get\_Path()** por el nombre de la página actual. Este método le permite usar el control de menú en cualquier página, sin importar cómo se llame. Para obtener más información, vea [HttpRequest.Path \(Propiedad\)](#).
- A continuación, el método **DataBinder.Eval** toma el número de índice del elemento y lo sustituye por el símbolo "{0}" en la cadena de formato. Para obtener más información, vea [DataBinder.Eval \(Método\)](#).

Por ejemplo, si la página se denomina MyPage.aspx, y el índice de este elemento es 1, la dirección URL generada sería la siguiente:

```
MyPage.aspx?category=1
```

## Exponer propiedades a la página de formularios Web Forms

Los controles **DataList** se pueden enlazar con varios tipos distintos de orígenes de datos. En aras de la simplicidad, en este tutorial se enlaza el control **DataList** con una matriz de cadenas simple, que se pasa de la página de formularios Web Forms al control del usuario mediante una propiedad expuesta por el control de usuario.

### Para declarar las propiedades de la matriz y la selección

1. Presione **F7** para cambiar de la vista Diseño al archivo de código fuente.
2. Agregue una declaración pública de nivel de clase para una matriz de cadenas y denomínela `values`.

Esta matriz contendrá la lista de los elementos de datos.

- De igual modo, declare la variable `selection` como **Integer** e inicialícela con el valor -1.

Esta variable contendrá el valor de índice del elemento de menú que seleccione el usuario. Como -1 no es un valor de índice en la matriz, no se le asignará un valor a la variable durante la inicialización.

La parte superior del código debería tener el siguiente aspecto:

```
// Visual J#
public abstract class menu extends System.Web.UI.UserControl
{
    protected System.Web.UI.WebControls.DataList DataList1;
    public System.String[] values;
    public int selection = -1;
```

## Inicializar el control

Ahora que las rutas y los formatos están establecidos, tiene que agregar el código para inicializar la conexión entre el control y los datos. Use el evento `Page_Load` para realizar esta inicialización.

- En el procedimiento **Page\_Load**, establezca `values` para que sea la propiedad **DataSource** del control **DataList**.
- Establezca **selection** como la propiedad **SelectedIndex** del control **DataList**.
- Invoque al método **DataBind()** del control **DataList** para crear los elementos de `DataList`.

El procedimiento **Page\_Load** debería tener este aspecto:

```
// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    DataList1.set_DataSource(values);
    DataList1.set_SelectedIndex(selection);
    DataList1.DataBind();
}
```

- Guarde y cierre el control.

## Utilizar el control de usuario

Ahora que ha creado un control de usuario, lo utilizará en la página de formularios Web Forms. Es posible usar el control en muchas páginas y enlazarlo con distintos datos de cada página. Sin embargo, en este tutorial, se usará el control de usuario en una única página denominada `MyPage.aspx`.

### Para agregar el control de usuario a una página de formularios Web Forms

- En el menú **Proyecto**, haga clic en **Agregar formulario Web**.

Aparecerá el cuadro de diálogo **Agregar nuevo elemento**, con **Web Forms** seleccionado.

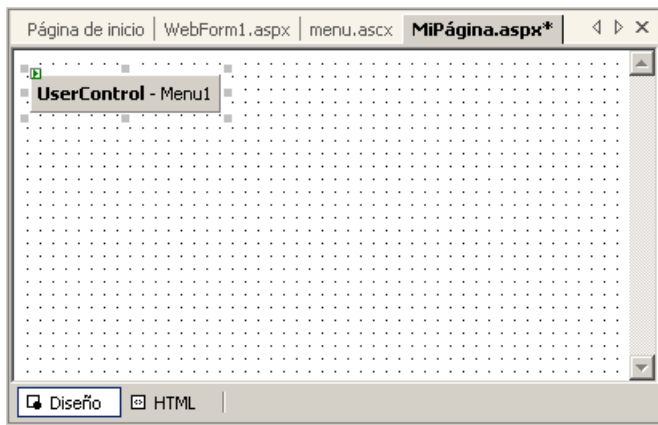
- Cambie el nombre por **MyPage.aspx** y haga clic en **Abrir**.

La nueva página de formularios Web Forms se abrirá en la vista Diseño.

- En el Explorador de soluciones, seleccione el archivo `menu.ascx` y arrástrelo a la página. Aparecerá como un control de usuario, con el ID `Menu1`.

El control de usuario aparecerá en la página como una etiqueta que muestra su tipo y nombre; no existe representación gráfica para los controles de usuario en las páginas ya que este tipo de control no está completamente compilado hasta que éstas no se ejecutan.

### Glifo de control de usuario en una página de formularios Web Forms



- De la ficha **Web Forms** del Cuadro de herramientas, agregue un control **Label** a la página.

Este control se usará para mostrar la información asociada con el elemento de menú seleccionado.

- Guarde los archivos.

Como Visual Studio no agrega automáticamente una declaración de controles de usuario al código, el paso siguiente es agregar la declaración manualmente.

#### Para agregar una declaración de código fuente para el control de usuario

- Presione **F7** para cambiar de la vista Diseño al archivo de código fuente.
- En el área de declaraciones, agregue una línea para declarar el control de usuario **Menu1** como menú.

```
// Visual J#
public class MyPage extends System.Web.UI.Page
{
    protected System.Web.UI.WebControls.Label Label1;
    protected menu Menu1;
```

Ahora que tiene una declaración de código para el control de usuario, deberá establecer sus propiedades mediante programa.

#### Para crear un origen de datos para el control de usuario

- En el método **Page\_Load** de la página de formularios Web Forms, declare una matriz de cadenas y llámela **values**. Ésta es la matriz que invocará el control de usuario que ha creado. Rellene la matriz con tres elementos: "News", "Events" y "Reference".
- Asigne la matriz a la propiedad `values` del control de usuario `Menu1`.

El código debería ser como el que sigue a continuación:

```
// Visual J#
//set datasource for menu
System.String[] values = new System.String[]{"News", "Events", "Reference"};
Menu1.values = values;
```

Para averiguar qué elemento del menú está seleccionado, marque el parámetro "category" que estableció cuando creó la expresión de enlace personalizada para el control **Hyperlink**.

#### Para capturar la selección del control de usuario

- En el método **Page\_Load**, agregue una línea de código para copiar la selección actual (si hay alguna) de la cadena de consulta de la dirección URL:

```
// Visual J#
String selectionId= get_Request().get_Params().get_Item("category");
```

- Si se estableció la selección, pásela al control de usuario mediante la propiedad `selection` del control y establezca el texto del control **Label** para que indique la selección actual. La página de formularios Web Forms puede usar la información de selección para filtrar la vista que muestra.



```
// Visual J#
if(!((selectionId == null)|| (selectionId == "")))
{
    int SelectIndex = (int) System.Int16.Parse(selectionId);
    Menu1.selection = SelectIndex;
    Label1.set_Text("Current selection is: " + values[SelectIndex]);
}
```

Esto completa la página de formularios Web Forms. El procedimiento **Page\_Load** completo debería tener este aspecto:

```
// Visual J#
System.String[] values = new System.String[]{"News", "Events", "Reference"};
Menu1.values = values;
String selectionId= get_Request().get_Params().get_Item("category");
if(!((selectionId == null)|| (selectionId == "")))
{
    int SelectIndex = (int) System.Int16.Parse(selectionId);
    Menu1.selection = SelectIndex;
    Label1.set_Text("Current selection is: " + values[SelectIndex]);
}
```

3. Guarde y cierre la página.

## Probar la página de formularios Web Forms

Ahora puede probar la página para ver la funcionalidad del menú.

### Para probar la página de formularios Web Forms

1. Haga clic con el botón secundario del *mouse* en MyPage.aspx en el Explorador de soluciones y seleccione **Establecer como página de inicio** en el menú contextual.
2. Presione F5 para ejecutar MyPage.aspx y probar el control de usuario. Haga clic en cada una de las opciones de menú para probar su funcionalidad.

El control de usuario muestra cada cadena que se pasa de la matriz en la página de formularios Web Forms como un vínculo independiente. La propiedad **DataSource** del control **DataList** está establecida en la matriz y se usa el control **DataList** para representar un hipervínculo para cada uno de los elementos de la propiedad **DataSource**. El control **DataList** también se ocupa de aplicar un estilo visual distinto al vínculo seleccionado.

### Vea también

[Desarrollar controles de servidor de ASP.NET | Plantilla de biblioteca de controles Web](#) |  
[Recomendaciones sobre los controles Web de usuario y los controles Web personalizados](#) |  
[Introducción a los controles de usuario Web](#) |  
[Tutorial: convertir una página de formularios Web Forms en un control de usuario con Visual J#](#) |  
[Tutoriales sobre los formularios Web Forms](#)

# Tutorial: convertir una página de formularios Web Forms en un control de usuario con Visual J#

Si ha desarrollado una página de formularios Web Forms y decide que le gustaría tener acceso a sus funciones en toda la aplicación, puede realizar algunos pequeños cambios en el archivo para convertirla en un control de usuario. Los controles de usuario Web son muy similares a las páginas de formularios Web Forms, y se crean usando la misma técnica. Cuando se convierte una página de formularios Web Forms en un control de usuario Web, se crea un componente de interfaz de usuario reutilizable en otras páginas de formularios Web Forms.

En este tutorial, creará una página de formularios Web Forms básica que presenta un mensaje de bienvenida personalizado al usuario. Para convertir una página en un control de usuario Web deberá realizar algunos cambios en el código:

- Cambie la clase base de código subyacente de **Page** a **UserControl**.
- Elimine las etiquetas `<html>`, `<head>`, `<body>` y `<form>` del archivo .aspx.
- Cambie el tipo de directiva ASP.NET de **@ Page** a **@ Control**.
- Cambie el atributo **Codebehind** para que haga referencia al archivo de clase de código subyacente del control (ascx.jsl).
- Cambie la extensión del archivo de .aspx a .ascx

Para probar el control, creará una nueva página de formularios Web Forms, agregará el control y abrirá la página en el explorador.

## Crear la página de formularios Web Forms

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:
  - a. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
  - b. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
  - c. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo `http://` y el nombre del servidor. La última parte de la dirección URL es el nombre del proyecto; en este tutorial, asígnele el nombre **Conversion**. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

**Sugerencia** Si ya tiene abierta una solución, seleccione **Cerrar solución** para que el nuevo proyecto de formularios Web Forms forme parte de su propia solución.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

Ahora puede agregar algunos controles y escribir código para mostrar un mensaje de bienvenida personalizado.

### Para agregar y programar controles

1. En la ficha **Web Forms** del Cuadro de herramientas, arrastre un objeto **TextBox** al diseñador.
2. Arrastre un control **Label** a la izquierda del cuadro de texto. En la ventana **Propiedades**, cambie la propiedad **Text** del control **Label** a **Escriba un nombre**.
3. Arrastre un control **Button** debajo del cuadro de texto y cambie su propiedad **Text** por **Entrar**.
4. Arrastre un segundo control **Label** debajo del botón y borre el texto predeterminado de forma que la propiedad **Text** aparezca vacía.
5. Haga doble clic en el control **Button** para abrir la vista **Código** con un controlador de eventos **Button\_Click** agregado.
6. Agregue el código siguiente al procedimiento **Button1\_Click**:

```
// Visual J#
Label2.set_Text("Hi " + TextBox1.get_Text() + ", welcome to Visual Studio .NET!");
```

No cierre el archivo porque volverá a utilizarlo posteriormente en el tutorial.

7. Presione CTRL+F5 para ejecutar la página de formularios Web Forms y probarla. Cuando termine, cierre el explorador que abrió cuando probó la página.

Ahora ya tiene una página de formularios Web Forms básica que puede convertir en un control de usuario Web.

## Convertir la página en un control de usuario

Como los elementos `<html>`, `<body>` y `<form>` no se incluyen en los controles de usuario, deberá borrarlos e incluirlos en la página de formularios Web Forms principal. También deberá cambiar el tipo de la directiva ASP.NET de la página de formularios Web Forms de `@ Page` a `@ Control`. Estas directivas ASP.NET especifican valores usados por la página y los compiladores de controles de usuario. Para obtener más información, vea [Sintaxis de directivas](#).

### Para convertir la página de formularios Web Forms en un control de usuario

1. Vuelva a WebForm1.aspx en el diseñador, y cambie a la vista HTML haciendo clic en la ficha HTML en la parte inferior de la ventana del diseñador.
2. Elimine las etiquetas `<html>`, `<!doctype>`, `<head>` (y su contenido), `<body>` y `<form>` del archivo .aspx. Si está utilizando Visual J#, el código HTML tendría este aspecto:

```
// Visual J#
<%@ Page language="VJ#" Codebehind="WebForm1.aspx.jsl" AutoEventWireup="false" Inherits="
WebApplication1.WebForm1" %>
<asp:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 168px; POSITION: absolute; TOP: 80p
x" runat="server"></asp:TextBox>
<asp:Label id="Label1" style="Z-INDEX: 102; LEFT: 56px; POSITION: absolute; TOP: 80px" ru
nat="server">Enter Name:</asp:Label>
<asp:Button id="Button1" style="Z-INDEX: 104; LEFT: 216px; POSITION: absolute; TOP: 136px
" runat="server" Text="Enter"></asp:Button>
<asp:Label id="Label2" style="Z-INDEX: 103; LEFT: 176px; POSITION: absolute; TOP: 200px"
runat="server"></asp:Label>
```

**Nota** El Editor de código agrega subrayados cuando encuentra problemas en el código. Estas marcas desaparecerán cuando termine la conversión.

3. Cambie el tipo de directiva ASP.NET de `@ Page` a `@ Control`. En un proyecto de Visual J#, tendrá el siguiente aspecto:

```
// Visual J#
<%@ Control Language="VJ#" Codebehind="WebForm1.aspx.jsl" AutoEventWireup="false"
Inherits="Conversion.WebForm1">
```

4. Cambie la referencia al atributo **Codebehind** de la directiva para que refleje el hecho de que la extensión .aspx se cambiará a .ascx, que es la extensión de los controles de usuario. Posteriormente, también cambiará el nombre del control a welcome.ascx; por tanto, establezca el atributo **Codebehind** en **welcome.ascx.jsl**.
5. De manera similar, cambie el atributo **Inherits** de la directiva para que haga referencia a `<nombre_de_proyecto>.welcome`.
6. Vuelva al archivo **WebForm1.aspx.jsl**.
7. Cambie la clase base de **System.Web.UI.Page** a **System.Web.UI.UserControl**, y cambie la declaración **Public Class** de **WebForm1** a **welcome**.

```
// Visual J#
public class welcome extends System.Web.UI.UserControl
```

8. Guarde y cierre el archivo de código subyacente y el archivo .aspx.
9. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en **WebForm1.aspx** y, a continuación, en el menú contextual, haga clic en **Cambiar nombre**.
10. Cambie el nombre del archivo por **welcome.ascx** para reflejar su nueva función. Es importante darse cuenta de que hay que cambiar la extensión del nombre de archivo de .aspx a .ascx. Cuando aparezca el cuadro de diálogo de confirmación, haga clic en **Sí**.

El archivo es ahora un control de usuario Web y está listo para usarlo en páginas de formularios Web Forms del proyecto.

## Probar el control de usuario

Puede probar el control de usuario creando una nueva página de formularios Web Forms y agregando el control.

### Para probar el control de usuario

1. En el menú **Proyecto**, haga clic en **Agregar formulario Web**. Aparecerá el cuadro de diálogo **Agregar nuevo elemento**, con **Web Forms** seleccionado.
2. Cambie el **Nombre** a WelcomeTest.aspx y haga clic en **Abrir**. La nueva página de formularios Web Forms se abrirá en el diseñador.
3. Arrastre welcome.ascx desde el Explorador de soluciones a la superficie de diseño.
4. Haga clic con el botón secundario del *mouse* en WelcomeTest.aspx en el Explorador de soluciones y haga clic en **Establecer como página de inicio** en el menú contextual.
5. Presione F5 para ejecutar la página de formularios Web Forms y probarla.

### Vea también

[Introducción a los controles de usuario Web](#) |

[Recomendaciones sobre los controles Web de usuario y los controles Web personalizados](#) |

[Desarrollar controles de usuario en un archivo de código subyacente](#) | [Tutoriales sobre los formularios Web Forms](#)

# Tutorial: crear un control Web personalizado con Visual J#

Si ninguno de los controles de servidor ASP.NET existente cumple los requisitos de la aplicación, se puede crear un control personalizado derivado de una de las clases de controles básicas. Estas clases proporcionan toda la funcionalidad básica de los controles de servidor, para que se pueda concentrar en programar las características necesarias.

En este tutorial, utilizará el código para un control Label personalizado incluido de manera predeterminada en la plantilla **Biblioteca de control Web**. Este control se deriva de la clase **WebControl** y se comporta como un control Label estándar, con la propiedad adicional de procesar el valor de la propiedad **Text** como un hipervínculo. Para completar el tutorial, realizará los siguientes procedimientos:

- Crear dos proyectos en la misma solución: uno para el control Label personalizado y el otro para que lo consuma la página de formularios Web Forms y probar el control.
- Agregar el control al Cuadro de herramientas.
- Consuma el control personalizado agregándolo a la página de formularios Web Forms.
- Crear un diseñador personalizado para el control.
- Probar el control en el explorador.

Este tutorial se centra no tanto en la escritura de código para el control, sino en qué hacer con un control una vez que el código está escrito.

## Crear los proyectos

El primer paso consiste en iniciar un nuevo proyecto con la plantilla de controles personalizados y generar el control.

### Para crear el control personalizado

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.  
Aparecerá el cuadro de diálogo **Nuevo proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**. En el panel **Plantillas**, seleccione **Biblioteca de controles Web**.
3. Cambie el **Nombre** por **CustomLabel** y haga clic en **Aceptar**.

Se creará el nuevo proyecto, y **WebCustomControl1** se abrirá en el Editor de código.

La plantilla **Biblioteca de controles Web** incluye código de un control **Label** personalizado de manera predeterminada. Éste es el control que usará en este tutorial.

4. En el menú **Generar**, haga clic en **Generar CustomLabel** para compilar el control.

El control se compila como CustomLabel.dll. De manera predeterminada, se crea en la carpeta **Bin**, dentro de la carpeta de proyecto **CustomLabel**.

5. Guarde el archivo.

Ahora que ya está creado el control, agregará una página de formularios Web Forms a la misma solución para que la página pueda probar el control.

### Para crear la página de formularios Web Forms

1. En el menú **Archivo**, elija **Agregar proyecto** y, después, haga clic en **Nuevo proyecto**.  
Aparecerá el cuadro de diálogo **Agregar nuevo proyecto**.
2. En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**. En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
3. En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo http:// y el nombre del servidor. La última parte de la dirección URL es el nombre del proyecto; en este tutorial, llámelo **LabelWebForm**. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

4. Haga clic en **Aceptar**.

Se creará el nuevo proyecto, y **WebForm1** se abrirá en el diseñador.

5. Guarde el proyecto.

## Agregar el control al Cuadro de herramientas

Ahora que ha compilado el control y la página de formularios Web Forms está lista para probarlo, puede agregar el control al Cuadro de herramientas para que esté disponible.

### Para agregar el control al Cuadro de herramientas

1. En el menú **Herramientas**, haga clic en **Agregar o quitar elementos del cuadro de herramientas**.
2. En la ficha **Componentes de .NET Framework** del cuadro de diálogo **Personalizar cuadro de herramientas**, haga clic en el botón **Examinar**. Busque **CustomLabel.dll**, selecciónelo y haga clic en **Abrir** para agregar **WebCustomControl1** a la lista de componentes del cuadro de diálogo **Personalizar cuadro de herramientas**.
3. En la lista de componentes de .NET Framework, seleccione **WebCustomControl1** y haga clic en **Aceptar**. **WebCustomControl1** se agregará al Cuadro de herramientas.

Cuando termine, debería ver un icono en el cuadro de herramientas con el siguiente aspecto:



El paso siguiente es probar el control en la página.

### Para agregar el control a la página de formularios Web Forms y probarlo

1. Abra **WebForm1** en la vista Diseño y arrastre **WebCustomControl1** del Cuadro de herramientas a la página. La representación predeterminada del control, que es simplemente su identificador, aparece en la vista Diseño.
2. Cambie a la vista HTML y compruebe que se ha agregado al código HTML de la página una directiva **@ Register** para el ensamblado del control, con el prefijo de etiqueta **TagPrefix** "cc1".
3. Establezca la propiedad **Text** del control en **Hello**.

La apariencia del control en la vista Diseño se actualizará para mostrar el nuevo texto cuando cambie a la vista Diseño.

En los pasos siguientes, va a personalizar el control, para lo que tendrá que quitarlo del Cuadro de herramientas.

### Para quitar el control del Cuadro de herramientas

1. En el Cuadro de herramientas, haga clic con el botón secundario del *mouse* en **WebCustomControl1** y, en el menú contextual, haga clic en **Eliminar**. Esta acción no suprimirá el control del proyecto, sólo del Cuadro de herramientas.
2. Cierre la página Web Forms sin guardarla.

El control Label personalizado funciona, pero no es lo suficientemente atractivo y la etiqueta no se distingue en el código. El siguiente paso es individualizar el control.

## Individualizar el control

El control está usando el icono predeterminado del Cuadro de herramientas y el prefijo de etiqueta predeterminado "cc1". Para hacer que el control sea más fácil de identificar, cambie el prefijo de etiqueta y cree un icono nuevo.

### Para cambiar el prefijo de etiqueta

1. Debajo del proyecto **CustomLabel** en el Explorador de soluciones, haga doble clic en **AssemblyInfo** para abrir el archivo AssemblyInfo en el diseñador. Agregue una instrucción import en la parte superior de la página AssemblyInfo:

```
// Visual J#  
import System.Web.UI.*;
```

2. Agregue un atributo **Assembly: TagPrefix** a la lista de atributos de ensamblados. Especificará el nuevo **TagPrefix** para el control **CustomLabel**, en este caso **xxx**.

```
// Visual J#  
/** @assembly System.Web.UI.TagPrefixAttribute("CustomLabel", "xxx") */
```

### Para crear un icono nuevo

1. En el Explorador de soluciones, seleccione el proyecto **CustomLabel**. En el menú **Proyecto**, haga clic en **Agregar nuevo elemento**.

Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

2. En el panel Plantillas, seleccione **Archivo de mapa de bits**, cambie el nombre por **WebCustomControl1.bmp**, y haga clic en **Abrir**. El nuevo archivo de mapa de bits aparecerá en el Explorador de soluciones y se abrirá en el diseñador.
3. En la ventana Propiedades, cambie las propiedades **Height** y **Width** del mapa de bits a 16. Los iconos deben tener un tamaño de 16 x 16 píxeles.
4. Dibuje una imagen distintiva en el mapa de bits usando varios colores, de forma que pueda reconocer el icono.
5. Guarde y cierre el archivo de mapa de bits.
6. Seleccione **WebCustomControl1.bmp** en el Explorador de soluciones. En la ventana Propiedades, cambie la propiedad **Generar acción** a **Recurso incrustado**.
7. En el Explorador de soluciones, seleccione el proyecto **CustomLabel**. En el menú **Generar**, haga clic en **Volver a generar CustomLabel**.
8. Guarde el trabajo.

Cuando agregue el control al Cuadro de herramientas, el icono creado aparecerá junto al nombre del control. Cuando utilice el control en la página de formularios Web Forms, la etiqueta HTML del control contendrá el prefijo identificativo que haya especificado.

### Para usar un control personalizado

1. Agregue **WebCustomControl1** al Cuadro de herramientas de nuevo.

Aparecerá el control con el icono creado. Dependiendo de la configuración, puede aparecer debajo de la ficha **General** del Cuadro de herramientas.

2. Vuelva a abrir la página de formularios Web Forms **WebForm1** y arrastre el control a dicha página.

**Nota** La propiedad **Text** se restablece al valor predeterminado porque cerró la página de formularios Web Forms sin guardar lo anterior.

3. Cambie a la vista HTML y compruebe que el atributo **TagPrefix** de la directiva **@ Register** es **xxx**.
4. Como va a modificar de nuevo el control, elimine **WebCustomControl1** de la página de formularios Web Forms, pero déjelo en el Cuadro de herramientas.

El control personalizado funciona y está individualizado, pero su aspecto en la superficie de diseño no es muy descriptivo. Para que el control tenga un mejor aspecto, le agregará un diseñador personalizado.

### Crear un diseñador personalizado

Los diseñadores son clases que permiten modificar la apariencia y el comportamiento de los componentes y los controles en tiempo de diseño. Aunque el objetivo habitual de cualquier diseñador de formularios WYSIWYG (lo que ve es lo que se imprime) consiste en reducir al mínimo las diferencias de presentación en tiempo de diseño y en tiempo de ejecución, en ocasiones se necesitan indicaciones especiales en tiempo de diseño. Para obtener más información, vea [Diseñadores personalizados](#).

En este tutorial, implementará un diseñador personalizado que haga que el control personalizado procese el texto como un hipervínculo.

### Para crear un diseñador personalizado para el control

1. En el Explorador de soluciones, seleccione el proyecto **CustomLabel**.
2. En el menú **Proyecto**, haga clic en **Agregar referencia** y agregue una referencia a **System.Design.dll**.
3. En el menú **Proyecto**, haga clic en **Agregar nuevo elemento**.

Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

4. En el panel Plantillas, seleccione **Clase**, cambie el nombre por **SampleDesigner** y haga clic en **Abrir**. El nuevo archivo de clase aparecerá en el Explorador de soluciones y se abrirá en el diseñador.
5. Agregue código que reemplace el método [GetDesignTimeHtml](#) de la clase base. A este método se llama en tiempo de diseño para obtener la representación del control. En el método sobrescrito, haga lo siguiente:
  - a. Obtenga la propiedad **Text** del control personalizado.
  - b. Si la propiedad **Text** ya se ha configurado (es decir, no es una cadena vacía), cree un nuevo control **Hyperlink**, establezca sus propiedades para que coincidan con el control original y llame al método **Render** del control **Hyperlink**.

El resultado final es que el control **Hyperlink** se muestra en el diseñador mediante el valor de la propiedad **Text**.

El código para realizar estos pasos tiene el siguiente aspecto, que muestra el archivo **SampleDesigner** completo:

```
// Visual J#
package CustomLabel;
import System.*;
import System.IO.*;
import System.Web.*;
import System.Web.UI.*;
import System.Web.UI.WebControls.*;
import System.Web.UI.Design.*;

// Summary description for SampleDesigner.
public class SampleDesigner extends System.Web.UI.Design.ControlDesigner
{
    public SampleDesigner()
    {
        //
        // TODO: Add Constructor Logic here
        //
    }
    public String GetDesignTimeHtml()
    {
        // Component is the control instance, defined in the base
        // designer
        WebCustomControl1 ctl = (WebCustomControl1 ) this.get_Component();

        if (ctl.get_Text() != "" && ctl.get_Text() != null)
        {
            StringWriter sw = new StringWriter();
            HtmlTextWriter tw = new HtmlTextWriter(sw);

            HyperLink placeholderLink = new HyperLink();

            // put control text into the link's Text
            placeholderLink.set_Text(ctl.get_Text());
            placeholderLink.set_NavigateUrl(ctl.get_Text());
            placeholderLink.RenderControl(tw);

            return sw.ToString();
        }
        else
            return GetEmptyDesignTimeHtml();
    }
}
```

6. Abra **WebCustomControl1.jsl** en el diseñador y agregue este atributo a la lista de atributos justo antes de la declaración de clase **WebCustomControl1**:

```
// Visual J#
/** @attribute DesignerAttribute("CustomLabel.SampleDesigner, CustomLabel")*/
```

La declaración de la clase **WebCustomControl1** debe ser similar a la siguiente:

```
// Visual J#
/** @attribute DesignerAttribute("CustomLabel.SampleDesigner, CustomLabel")*/
/**
```



```
* @attribute DefaultValue("Text")
* @attribute ToolboxData("<{0}:WebCustomControl1 runat=server> </{0}:WebCustomControl1>
")
*/
public class WebCustomControl1 extends System.Web.UI.WebControls.WebControl
```

7. En el menú **Generar**, haga clic en **Generar solución** para generar ambos proyectos.
8. Guarde el trabajo y cierre todos los archivos abiertos.

## Probar el control

### Para probar que la propiedad Text funciona correctamente

1. Abra **WebForm1** en el diseñador.
2. Arrastre **WebCustomControl1** desde el Cuadro de herramientas a la página.
3. Establezca la propiedad **Text** del control Web personalizado en **Hello**. El texto debería aparecer como hipervínculo en la página de formularios Web Forms.

### Para probar el control en un explorador

1. En el Explorador de soluciones, seleccione **WebForm1**. En el menú **Archivo**, haga clic en **Ver en el explorador**.
2. Compruebe que aparece **Hello** en el explorador. El texto tiene el aspecto de una etiqueta, no de un hipervínculo.

**Nota** En tiempo de diseño en Visual Studio, el código ejecutado siempre es de confianza, aunque posteriormente se encuentre en un proyecto en el que el código no sea de confianza en tiempo de ejecución. Es decir, un control personalizado podría funcionar perfectamente durante las pruebas en el equipo del programador, pero podría dar errores a causa de la falta de los permisos adecuados en una aplicación implementada. Compruebe siempre los controles en el contexto de seguridad donde se ejecutarán en aplicaciones reales.

## Vea también

[Desarrollar controles de servidor de ASP.NET](#) |

[Recomendaciones sobre los controles Web de usuario y los controles Web personalizados](#) |

[Tutoriales sobre los formularios Web Forms](#)

# Tutorial: mostrar un documento XML en una página de formularios Web Forms mediante transformaciones con Visual J#

Como cada vez es más frecuente encontrar información en formato de documento XML, es muy posible que necesite trabajar con este tipo de documentos. En este tutorial se explica cómo puede mostrar fácilmente en una página de formularios Web Forms información de un documento XML "no estructurado", es decir, un documento XML que no representa datos relacionales.

## XML y XSL en páginas de formularios Web Forms

La información de un archivo XML es información "sin formato", que sólo contiene datos y no indica cómo darles formato, cómo mostrarlos, etc. Para mostrar la información XML en una página de formularios Web Forms, debe proporcionar información sobre el formato y la presentación. Por ejemplo, debe incluir las etiquetas <TABLE>, las etiquetas <P> o todo lo que necesite utilizar para mostrar la información. También debe incluir instrucciones que indiquen cómo los datos del archivo XML llenan estas etiquetas. Por ejemplo, ¿debe mostrarse cada elemento del archivo XML como una fila de una tabla?

Una forma de proporcionar estas instrucciones es crear una página de formularios Web Forms y después utilizar código para analizar el archivo XML y colocar los datos en las etiquetas HTML correspondientes. Sin embargo, es una forma tediosa y poco flexible. (Por otro lado, es muy eficaz porque proporciona un control preciso sobre el archivo XML mediante programación).

## Transformaciones XSL

Es mejor utilizar el lenguaje de transformación XSL y crear "transformaciones" o archivos XSL. Una transformación XSL contiene la siguiente información:

- Una plantilla, básicamente, una página HTML con las etiquetas apropiadas, que especifica cómo presentar la información XML.
- Instrucciones de procesamiento XSL, que especifican cómo ajustar la información del archivo XML en la plantilla.

Puede definir varias transformaciones y aplicarlas al mismo archivo XML. Así puede utilizar la información XML de distintas maneras, mostrarla de manera diferente, seleccionar distintos datos del archivo XML, etc.

Una vez creadas las transformaciones XSL, debe aplicarlas al archivo XML, es decir, el archivo XML se procesa transformándolo de acuerdo a uno de los archivos XSL. El resultado es un nuevo archivo con la información XML con un formato conforme al archivo de transformación.

Este proceso también se puede realizar mediante programación. No obstante, también puede obtener buenos resultados con el control de servidor Web **XML**. (Para ver una introducción, vea [XML \(Control de servidor Web\)](#).) Este control funciona como un marcador de posición en la página de formularios Web Forms. Puede establecer sus propiedades a un archivo XML y a una transformación XSL específicos y, cuando se procesa la página, el control lee el archivo XML, aplica la transformación y muestra los resultados.

Para obtener más información sobre XSL, vea [Transformaciones XSLT con la clase XslTransform](#).

## Objetivos del tutorial

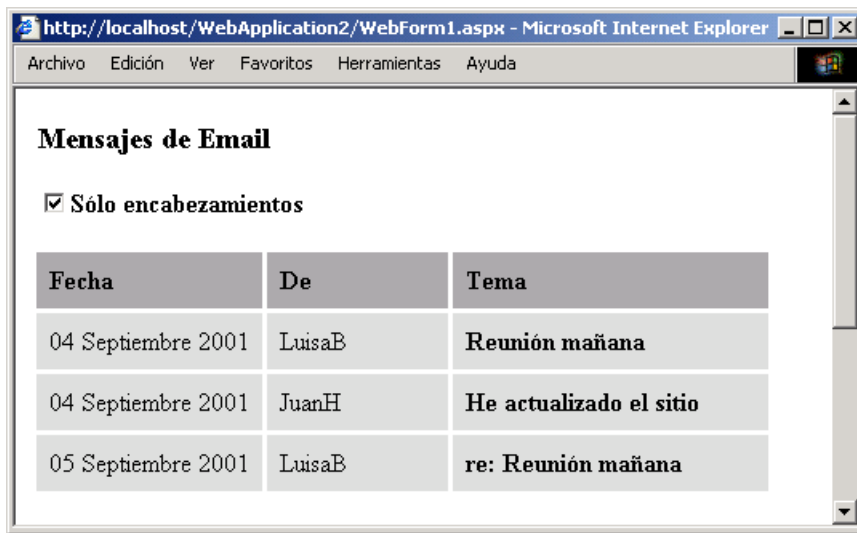
En este tutorial se explica cómo utilizar el control de servidor Web **XML** para mostrar información XML mediante transformaciones XSL. El escenario es sencillo. Tendrá lo siguiente:

- Un archivo XML que contiene varios mensajes de correo electrónico ficticios.
- Dos transformaciones XSL. Una muestra sólo la fecha, el remitente y el asunto del mensaje de correo. La otra muestra el mensaje de correo completo.

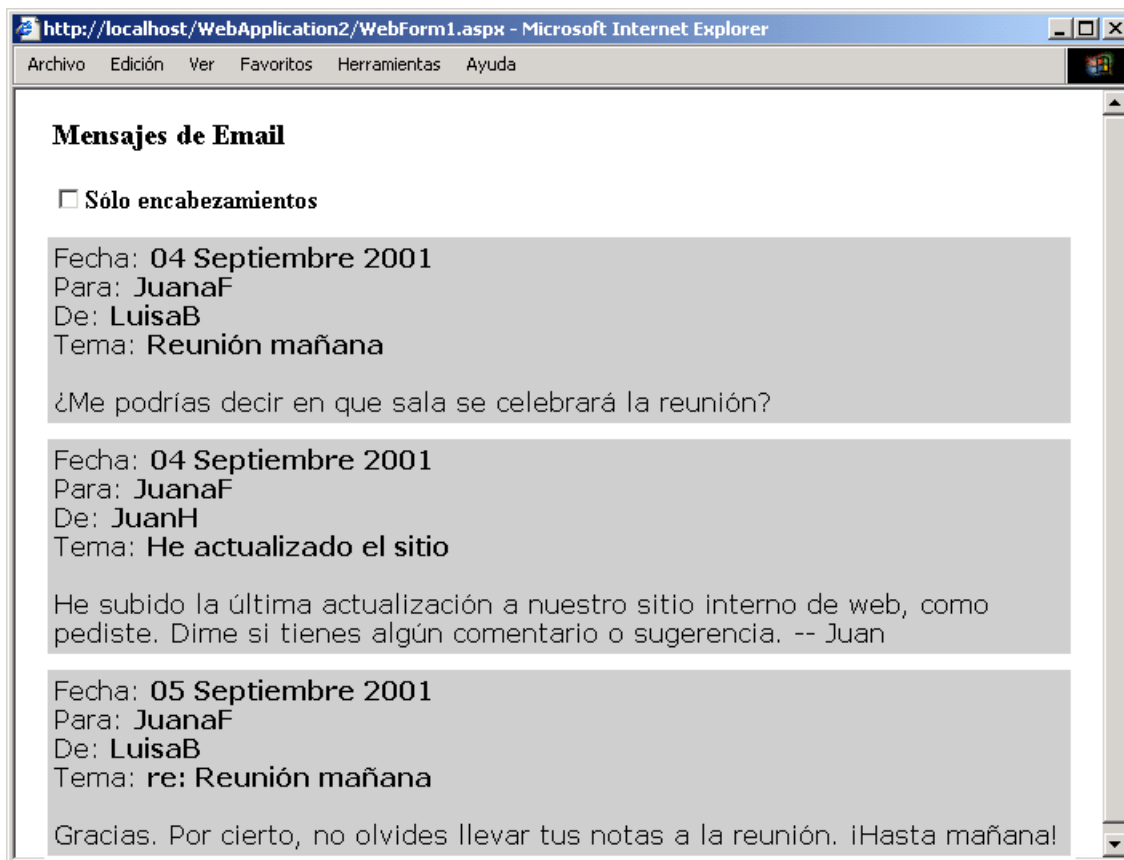
**Nota** Todos estos archivos forman parte del tutorial.

Crearé una página de formularios Web Forms que permita a los usuarios mostrar la información XML de dos maneras diferentes. La página contiene una casilla de verificación "Sólo encabezados" que está activada de manera predeterminada. Por lo tanto, la transformación predeterminada es aquella que sólo muestra los encabezados del mensaje de correo. Si el usuario desactiva la casilla, se vuelve a mostrar la información XML con los mensajes de correo completos.

Cuando la página sólo muestra los encabezados, tiene un aspecto similar al siguiente:



Cuando el usuario desactiva la casilla de verificación, el aspecto de la página es:



Para completar este tutorial, debe disponer de los permisos necesarios para crear proyectos de aplicación Web ASP.NET en el equipo donde se encuentre el servidor Web.

Como parte del tutorial, realizará lo siguiente:

- Creará un nuevo proyecto de aplicación Web y una página de formularios Web Forms.
- Agregará al proyecto el archivo XML y los dos archivos de transformación XSL.
- Agregará un control de servidor Web **XML** a la página de formularios Web Forms y lo configurará para que muestre el archivo XML y una transformación.
- Agregará controles y código que permitan al usuario cambiar la transformación aplicada al archivo XML.

## Crear el proyecto y el formulario

El primer paso es crear una aplicación Web y una página de formularios Web Forms.

### Para crear el proyecto y el formulario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, haga lo siguiente:

- En el panel **Tipos de proyecto**, elija **Proyectos de Visual J#**.
- En el panel **Plantillas**, seleccione **Aplicación Web ASP.NET**.
- En el cuadro **Ubicación**, escriba la dirección URL completa de la aplicación, incluyendo http://, el nombre del servidor y el nombre del proyecto. El servidor Web debe tener instalado IIS 5 o una versión posterior y .NET Framework. Si tiene instalado IIS en el equipo, puede especificar `http://localhost` para el servidor.

Cuando hace clic en **Aceptar**, se crea un nuevo proyecto de formularios Web Forms en el directorio raíz del servidor Web especificado. Además, se muestra una nueva página de formularios Web Forms denominada WebForm1.aspx en la vista Diseño del Diseñador de Web Forms.

**Sugerencia** Si existen problemas al crear el proyecto de aplicación Web, vea [Error de acceso al Web \(Cuadro de diálogo\)](#).

## Agregar el archivo XML y las transformaciones XSL

El escenario del tutorial asume que el archivo XML y los dos archivos de transformaciones XSL existen. En una aplicación real, podría crearlos u obtenerlos de algún otro sitio (por ejemplo, de otra aplicación). Para evitar tener que crear u obtener los archivos, se incluyen como parte del tutorial. Los pasos siguientes (que podrían no formar parte de una aplicación real) le permiten disponer de los archivos para que pueda continuar con el tutorial.

### Para agregar el archivo XML al proyecto

- En el Explorador de soluciones, haga clic con el botón secundario en el nombre del proyecto, elija **Agregar** y, a continuación, elija **Agregar nuevo elemento**.

Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

- En el panel **Plantillas**, seleccione **Archivo XML**.
- En el cuadro **Nombre**, cambie el nombre del archivo XML a **Emails.xml**.

**Nota** Puede asignar el nombre que desee al archivo XML. Si utiliza un nombre diferente, anótelos para utilizarlo posteriormente.

Se abre el editor XML con un nuevo archivo XML que contiene una directiva XML en la parte superior.

- Seleccione la directiva XML (y todo lo que contenga el archivo) y elimínela.
- Copie la siguiente información XML y péguela en el archivo utilizando el comando **Pegar como HTML**:

```
<?xml version="1.0" ?>
<MESSAGES>
  <MESSAGE id="101">
    <TO>JoannaF</TO>
    <FROM>LindaB</FROM>
    <DATE>04 September 2001</DATE>
    <SUBJECT>Meeting tomorrow</SUBJECT>
    <BODY>Can you tell me what room the committee meeting will be in?</BODY>
  </MESSAGE>
  <MESSAGE id="109">
    <TO>JoannaF</TO>
    <FROM>JohnH</FROM>
    <DATE>04 September 2001</DATE>
    <SUBJECT>I updated the site</SUBJECT>
    <BODY>I've posted the latest updates to our internal web site, as you requested. Let me know if you have any comments or questions. -- John
  </BODY>
  </MESSAGE>
  <MESSAGE id="123">
    <TO>JoannaF</TO>
    <FROM>LindaB</FROM>
    <DATE>05 September 2001</DATE>
    <SUBJECT>re: Meeting tomorrow</SUBJECT>
    <BODY>Thanks. By the way, do not forget to bring your notes from the conference. See you later!</BODY>
  </MESSAGE>
</MESSAGES>
```

```
</MESSAGE>
</MESSAGES>
```

6. Guarde el archivo y ciérrelo.

A continuación, agregará dos transformaciones XSL al proyecto.

### Para agregar las transformaciones XSL al proyecto

1. En el Explorador de soluciones, haga clic con el botón secundario en el nombre del proyecto, elija **Agregar** y, a continuación, elija **Agregar nuevo elemento**.

Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

2. En el panel **Plantillas**, seleccione **Archivo XSLT**.
3. En el cuadro **Nombre**, cambie el nombre del archivo a **Email\_headers.xslt**.

**Nota** Puede asignar el nombre que desee al archivo XML. Si utiliza un nombre diferente, anótelos para utilizarlo posteriormente.

Se abre el editor XML conteniendo algunas directivas XSLT y elementos.

4. Seleccione todo y elimínelo.
5. Copie lo siguiente y péguelo en el archivo utilizando el comando **Pegar como HTML**:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<HTML>
<BODY>
<TABLE cellspacing="3" cellpadding="8">
  <TR bgcolor="#AAAAAA">
    <TD class="heading"><B>Date</B></TD>
    <TD class="heading"><B>From</B></TD>
    <TD class="heading"><B>Subject</B></TD>
  </TR>
  <xsl:for-each select="MESSAGES/MESSAGE">
    <TR bgcolor="#DDDDDD">
      <TD width="25%" valign="top">
        <xsl:value-of select="DATE"/>
      </TD>
      <TD width="20%" valign="top">
        <xsl:value-of select="FROM"/>
      </TD>
      <TD width="55%" valign="top">
        <B><xsl:value-of select="SUBJECT"/></B>
      </TD>
    </TR>
  </xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

**Nota** Puede ajustar la plantilla como desee; por ejemplo, puede cambiar los colores, el tamaño de fuente, los estilos y otras características.

6. Guarde el archivo y ciérrelo.
7. Repita los pasos que van del 1 al 4, pero en este caso póngale por nombre al archivo **Email\_all.xslt**.
8. Pegue lo siguiente en el archivo **Email\_all.xslt** utilizando el comando **Pegar como HTML**:

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<HTML>
<BODY>
<FONT face="Verdana" size="2">
<TABLE cellpadding="10" cellspacing="4">
  <xsl:for-each select="MESSAGES/MESSAGE">
    <TR bgcolor="#CCCCCC">
      <TD class="info">
        Date: <B><xsl:value-of select="DATE"/></B><BR></BR>
        To: <B><xsl:value-of select="TO"/></B><BR></BR>
        From: <B><xsl:value-of select="FROM"/></B><BR></BR>
        Subject: <B><xsl:value-of select="SUBJECT"/></B><BR></BR>
        <BR></BR><xsl:value-of select="BODY"/>
      </TD>
    </TR>
  </xsl:for-each>
</TABLE>
</FONT>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

**Nota** Puede ajustar la plantilla como desee; por ejemplo, puede cambiar los colores, el tamaño de fuente, los estilos y otras características.

9. Guarde el archivo y ciérrelo.

## Agregar el control XML a la página de formularios Web Forms

Ahora que ya tiene un archivo XML y dos transformaciones, puede utilizarlos para mostrar información en una página de formularios Web Forms. Para ello utilizará un control de servidor Web **XML**.

### Para agregar un control XML

1. Abra o vaya a la página de formularios Web Forms creada de manera predeterminada cuando creó el proyecto de aplicación Web.
2. Desde la ficha **Web Forms** del Cuadro de herramientas, arrastre un control **XML** a la página.
3. En la ventana Propiedades, seleccione el control y establezca las propiedades siguientes:

Propiedad	Valor
<b>DocumentSource</b>	Emails.xml
<b>TransformSource</b>	Email_headers.xslt

Esto hace que el control **XML** muestre sólo los encabezados de los mensajes de correo de manera predeterminada.

## Agregar controles para cambiar las transformaciones

En este tutorial, utilizará una casilla de verificación que permita a los usuarios cambiar entre transformaciones. De manera predeterminada, el control **XML** aplica una transformación que sólo muestra los encabezados de los mensajes de correo.

### Para agregar una casilla de verificación para aplicar otra transformación

1. Desde la ficha **Web Forms** del **Cuadro de herramientas**, arrastre un control **CheckBox** al formulario.
2. Establezca las siguientes propiedades del control **CheckBox**:

Propiedad	Valor
<b>Text</b>	Sólo encabezados
<b>Checked</b>	<b>True</b>

<b>AutoPostBack</b>	<b>True.</b> Así el formulario se envía y procesa tan pronto como el usuario haga clic en la casilla.
---------------------	---

- Haga doble clic en la casilla de verificación.
- Se abre el Editor de código con un esqueleto del controlador del evento **CheckChanged** del control **CheckBox**.
- Agregue código que cambie entre las transformaciones **Email\_headers.xslt** y **Email\_all.xslt** dependiendo del estado de la casilla de verificación. El controlador **CheckChanged** completo tendría el siguiente aspecto:

```
// Visual J#
private void CheckBox1_CheckedChanged (Object sender, System.EventArgs e)
{
    if (CheckBox1.get_Checked())
    {
        Xml1.set_TransformSource("email_headers.xslt");
    }
    else
    {
        Xml1.set_TransformSource("email_all.xslt");
    }
}
```

## Pruebas

Ahora puede ver las transformaciones en ejecución.

### Para probar las transformaciones

- Presione F5 para ejecutar la página del formulario Web Forms.  
De manera predeterminada, verá la fecha, el remitente y la línea de asunto de los mensajes de correo.
- Desactive la casilla de verificación **Sólo encabezados**.  
Se vuelven a mostrar los mensajes de correo, pero esta vez con un diseño diferente y con el texto completo.

## Pasos siguientes

En este tutorial sólo se han explicado las bases para trabajar con un documento XML y con transformaciones. En una aplicación real, probablemente necesitará trabajar con el documento XML en mayor profundidad. A continuación se muestran algunas sugerencias para una mayor investigación:

- Crear transformaciones más sofisticadas** En este tutorial, sólo ha visto un ejemplo de cómo puede utilizar las transformaciones. XSL es un lenguaje eficaz, con una compatibilidad sofisticada que le permite no sólo crear páginas HTML sino también realizar cualquier tipo de transformación virtual de XML a otra estructura. Para obtener información detallada, vea el Manual del programador de XSLT de Microsoft.
- Aplicar transformaciones mediante programación** En este tutorial, el control de servidor Web **XML** realiza todo el trabajo de aplicar las transformaciones a los datos XML sin formato. Puede encontrar de utilidad para su aplicación realizar este trabajo usted mismo (quizás porque su aplicación no pueda utilizar un control **XML**). Para obtener información detallada, vea [Transformaciones XSLT con la clase XslTransform](#).
- Escribir documentos XML (en lugar de leerlos)** Con el control **XML** es muy fácil mostrar el contenido de un archivo XML en una página de formularios Web Forms. Sin embargo, es posible que desee crear o modificar usted mismo archivos XML. Para obtener más información, vea [Utilizar XML en .NET Framework](#).

## Vea también

[XML \(Control de servidor Web\)](#) | [Tutoriales sobre los formularios Web Forms](#)

# Tutorial: crear un servicio Web XML con Visual J#

El siguiente tutorial explica el proceso para escribir un servicio Web XML sencillo utilizando Visual J#.

Un servicio Web XML es un medio de obtener acceso a la funcionalidad del servidor en modo remoto. Con los servicios Web XML, las empresas pueden exponer interfaces de programación a sus datos o lógica empresarial que, a su vez, pueden obtener y manipular las aplicaciones de cliente y servidor. Los servicios Web XML permiten el intercambio de datos en escenarios cliente-servidor o servidor-servidor, utilizando estándares como los servicios de mensajería HTTP y XML para que los datos pasen los servidores de seguridad.

En WebService (Ejemplo, Generar un servicio Web XML) podrá ver un ejemplo más complejo. Para obtener más información sobre servicios Web XML, vea [Introducción a aplicaciones Web ASP.NET en Visual Studio](#) en Visual Studio .NET (MSDN).

## Para crear un servicio Web XML utilizando Visual J#

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, elija **Proyecto**.

Aparecerá el cuadro de diálogo **Nuevo proyecto**.

2. Seleccione la carpeta **Proyectos de Visual J#** y elija el icono **Servicio Web ASP.NET**.
3. En el cuadro **Ubicación**, escriba la dirección del servidor Web en el que desarrollará el servicio Web XML y especifique **VJSharpWebService** como nombre de directorio. De manera predeterminada, el proyecto utiliza su equipo local, "http://localhost".
4. Haga clic en **Aceptar**.

La plantilla de proyecto creará una solución y abrirá el Diseñador de componentes. Se creará el archivo de servicio Web Service1.asmx y el archivo de código subyacente correspondiente Service1.asmx.jsl.

5. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en Service1.asmx y elija **Ver código**.  
El Editor de código abrirá el archivo de código subyacente Service1.asmx.jsl.
6. Quite los comentarios del método **HelloWorld**.

El método resultante presenta el siguiente aspecto:

```
/** @attribute WebMethod() */  
public String HelloWorld()  
{  
    return "Hello World";  
}
```

Observe el atributo **WebMethod**, que indica que el método **HelloWorld** se va a exponer como un método de servicios Web.

7. Presione F5 para ejecutar el proyecto.

Se generará y ejecutará el proyecto. Internet Explorer abrirá el identificador URI <http://localhost/VJSharpWebService/Service1.asmx>. Ésta es la página de prueba para todos los métodos de servicios Web del servicio Web.

8. Haga clic en el hipervínculo **HelloWorld**.

Internet Explorer abrirá <http://localhost/VJSharpWebService/Service1.asmx?op=HelloWorld>. Ésta es la página de prueba del método de servicios Web **HelloWorld**.

9. Haga clic en el botón **Invocar**.

Internet Explorer abrirá una página nueva que muestre el código XML generado por el método de servicios Web **HelloWorld**, como se indica a continuación:

```
<?xml version="1.0" encoding="utf-8" ?>  
  <string xmlns="http://tempuri.org/">Hello World</string>
```



Para crear un cliente que tenga acceso a este servicio Web XML, vea uno de los siguientes tutoriales:

- [Tutorial: acceso a un servicio Web XML con un cliente de formularios Web Forms de Visual J#](#)
- [Tutorial: acceso a un servicio Web XML con un cliente de Windows de Visual J#](#)

### **Vea también**

[Tutoriales sobre servicios Web XML](#)

# Tutorial: acceso a un servicio Web XML con un cliente de formularios Web Forms de Visual J#

El siguiente tutorial explica el proceso para escribir una aplicación de formularios Web Forms sencilla utilizando Visual J#. La aplicación es un cliente para el servicio Web XML creado en [Tutorial: crear un servicio Web XML con Visual J#](#). En este tutorial se supone que se ha realizado el tutorial del servicio Web XML completo.

Los formularios Web Forms son componentes de ASP.NET que permiten crear formularios en páginas Web que pueden mostrar información en cualquier explorador, utilizando cualquier lenguaje de marcado, y utilizan código del servidor para implementar la lógica de la aplicación. Para obtener más información, vea [Crear y administrar formularios Web Forms](#) en Visual Studio .NET (MSDN).

En la primera parte de este tutorial, creará un cliente de Web Forms.

## Para crear un cliente de Web Forms utilizando Visual J#

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, elija **Proyecto**.

Aparecerá el cuadro de diálogo **Nuevo proyecto**.

2. Seleccione la carpeta **Proyectos de Visual J#** y elija el icono **Aplicación Web ASP.NET**.
3. En el cuadro **Ubicación**, escriba la dirección del servidor Web en el que desarrollará el servicio Web XML y especifique **VJSharpWebForms** como nombre de directorio. De manera predeterminada, el proyecto utiliza su equipo local, "http://localhost".
4. Haga clic en **Aceptar**.

La plantilla de proyecto creará una solución y abrirá el Diseñador de Web Forms. La plantilla de proyecto agrega un archivo de plantilla (WebForm1.aspx) al proyecto y abre este archivo en la vista Diseño. La plantilla predeterminada contiene un control de formulario.

La plantilla de proyecto crea también el archivo de código subyacente WebForm1.aspx.jsl. Este archivo tiene una clase, **WebForm1**, que extiende **System.Web.UI.Page**. El diseñador generará código en este archivo basado en los controles que agregue a la superficie de diseño.

5. Abra el Cuadro de herramientas, haga clic en la ficha **Web Forms** y arrastre un control de etiqueta y otro de botón hasta la superficie de diseño.

Se crearán los controles **Label1** y **Button1**.

6. Cambie la propiedad de texto de **Button1** para que diga **Haga clic aquí**.
7. Haga doble clic en **Button1**.

Se creará el controlador de eventos **Button1\_Click** en la clase **WebForm1**. Se abrirá el Editor de código.

8. Reemplace el cuerpo del controlador de eventos con el siguiente código:

```
Label1.set_Text ("Hello World");
```

9. Presione F5 para ejecutar el proyecto.

El proyecto se generará e Internet Explorer abrirá la página Web resultante con este identificador URI:

```
http://localhost/VJSharpWebForms/WebForm1.aspx
```

10. Haga clic en **Button1** en el formulario.

El texto **Label1** cambia a "Hello World".

En esta parte del tutorial, utilizará el servicio Web XML creado en [Tutorial: crear un servicio Web XML con Visual J#](#).

## Para utilizar el servicio Web XML

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo References del proyecto VJSharpWebForms y, a continuación, haga clic en **Agregar referencia Web**.

Se abrirá el cuadro de diálogo **Agregar referencia Web**.

2. Escriba el siguiente identificador URI de servicio Web en el cuadro **Dirección** y presione ENTRAR:

```
http://localhost/VJSharpWebService/Service1.asmx
```

Éste es el identificador URI del servicio Web que creó en [Tutorial: crear un servicio Web XML con Visual J#](#). El método de servicios Web **HelloWorld** aparecerá en el panel izquierdo del cuadro de diálogo **Agregar referencia Web**.

3. Haga clic en **Agregar referencia**.

Se generará una clase de proxy para el servicio Web VJSharpWebService y se agregará al proyecto. Ahora se puede invocar a los métodos de servicios Web como si se hubiesen definido en modo local. La clase de proxy enruta las llamadas a métodos de servicios Web hacia el servidor de servicios Web, utilizando SOAP con transporte HTTP.

4. Haga clic en la ficha **WebForm1.aspx** para volver a la superficie de diseño.
5. Abra el Cuadro de herramientas, haga clic en la ficha **Web Forms** y arrastre otra etiqueta de Web Forms a la superficie de diseño.

Se creará el control **Label2**.

6. Haga doble clic en **Button1**.

Se abrirá el Editor de código.

7. Agregue estas líneas al cuerpo del controlador de eventos **Button1\_Click**:

```
// Visual J#  
localhost.Service1 service = new localhost.Service1();  
Label2.set_Text(service.HelloWorld());
```

8. Presione F5 para ejecutar el proyecto.

El proyecto se generará e Internet Explorer abrirá la página Web resultante.

9. Haga clic en **Button1** en el formulario.

**Label1** y **Label2** cambian a "Hello World".

## Vea también

[Tutoriales sobre servicios Web XML](#) | [Tutorial: acceso a un servicio Web XML con un cliente de Windows de Visual J#](#)

# Tutorial: acceso a un servicio Web XML con un cliente de Windows de Visual J#

El siguiente tutorial explica el proceso para escribir una aplicación de formulario Windows Forms sencilla utilizando Visual J#. La aplicación es un cliente para el servicio Web XML creado en [Tutorial: crear un servicio Web XML con Visual J#](#). En este tutorial se supone que se ha realizado el tutorial del servicio Web XML completo.

En primer lugar, cree el cliente y agregue una referencia Web al servicio Web XML.

## Para crear una aplicación Windows con acceso a un cliente de servicio Web XML, utilizando Visual J#

1. En el menú **Archivo**, seleccione **Abrir solución**.

Aparecerá el cuadro de diálogo **Abrir solución**.

2. Seleccione la solución VJSharpWebService y haga clic en **Abrir**.
3. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en la raíz VJSharpWebService, elija **Agregar** y **Nuevo proyecto**.

Aparecerá el cuadro de diálogo **Nuevo proyecto**.

4. Seleccione la carpeta **Proyectos de Visual J#** y elija el icono **Aplicación para Windows**.
5. En el cuadro **Nombre**, escriba **VJSharpWinClient**.
6. Haga clic en **Aceptar**.

La plantilla de proyecto creará el proyecto VJSharpWinClient y abrirá el Diseñador de Windows Forms.

7. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo de proyecto VJSharpWinClient y elija **Establecer como proyecto de inicio**.
8. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo References del proyecto VJSharpWinClient y elija **Agregar referencia Web**.

Se abrirá el cuadro de diálogo **Agregar referencia Web**.

9. Escriba el siguiente identificador URI de servicio Web en el cuadro **Dirección** y presione ENTRAR:

```
http://localhost/VJSharpWebService/Service1.asmx
```

Éste es el identificador URI del servicio Web que creó en [Tutorial: crear un servicio Web XML con Visual J#](#). El método de servicio Web **HelloWorld** aparecerá en el panel izquierdo del cuadro de diálogo **Agregar referencia Web**.

10. Haga clic en **Agregar referencia**.

Se generará una clase de proxy para el servicio Web VJSharpWebService y se agregará al proyecto. Ahora se puede invocar a los métodos de servicios Web como si se hubiesen definido en modo local. La clase de proxy enruta las llamadas a métodos de servicios Web hacia el servidor de servicios Web, utilizando SOAP con transporte HTTP.

La siguiente tarea consiste en modificar la interfaz de usuario del cliente para que éste pueda invocar al servicio Web XML. A continuación, puede generar y probar el cliente.

## Para modificar la interfaz de usuario del cliente

1. Abra el Cuadro de herramientas y arrastre un control de etiqueta y otro de botón hasta Form1.

Se crearán los controles **label1** y **button1**.

2. Cambie la propiedad de texto de **button1** para que diga **Haga clic aquí**.
3. Haga doble clic en **button1**.

Se creará el controlador de eventos **button1\_Click** en el archivo de código fuente Form1.jsl. Se abrirá el Editor de código.

4. Reemplace el cuerpo del controlador de eventos con el siguiente código:

```
// Visual J#
localhost.Service1 service = new localhost.Service1();
label1.set_Text(service.HelloWorld());
```

---

5. Presione F5 para ejecutar el proyecto.

Se generará y ejecutará el proyecto.

6. Haga clic en **button1** en el formulario.

El texto **label1** cambiará a "Hello World".

### **Vea también**

[Tutoriales sobre servicios Web XML](#) |

[Tutorial: acceso a un servicio Web XML con un cliente de formularios Web Forms de Visual J#](#)

# Tutoriales sobre la creación de componentes y controles

Los siguientes tutoriales le orientarán en el uso de componentes, controles y subprocesamiento.

## En esta sección

### [Crear un componente con Visual J#](#)

Muestra cómo programar en J# un componente sencillo. Ilustra la interacción entre cliente y componente, el período de duración de un objeto y las referencias circulares, la depuración de clientes y componentes, y el uso de métodos compartidos y métodos de instancia.

### [Crear un componente sencillo multiproceso con Visual J#](#)

Muestra la creación de un componente multiproceso, explicando cómo funcionan los subprocesos y cómo coordinar varios subprocesos en un componente.

### [Crear un control de usuario con Visual J#](#)

Muestra el desarrollo de un sencillo control de usuario que se hereda de la clase **UserControl** y cómo heredar de un control de usuario establecido.

### [Crear una clase de colección propia con Visual J#](#)

Proporciona instrucciones detalladas para implementar su propia clase de colección con establecimiento inflexible de tipos.

### [Heredar de un control de Windows Forms con Visual J#](#)

Muestra cómo crear un sencillo control de botón heredado. Este botón hereda la funcionalidad del botón estándar de Windows Forms y expone un miembro personalizado.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, componentes y controles, trabajar con Windows Forms y Web Forms, y desarrollar tareas relacionadas con datos.

### [Tutoriales sobre el componente de servicios Framework](#)

Temas paso a paso que muestran varios elementos integrados en Windows, tales como registros de eventos, contadores de rendimiento y servicios.

### [Programar con componentes](#)

Sección de Visual Basic/Visual C# que explica los componentes de todos los tipos, incluidos los utilizados en servidores.

# Tutorial: crear un componente con Visual J#

Los componentes proporcionan código reutilizable en forma de objetos. Una aplicación que utiliza el código de un componente para crear objetos y llamar a sus métodos y propiedades se conoce como cliente. Un cliente puede estar o no en el mismo ensamblado que el componente que utiliza.

Los procedimientos se basan los unos en los otros, así que el orden en que se ejecutan es importante.

## Crear el proyecto

### Para crear la biblioteca de clases CDemoLib y el componente CDemo

1. En el menú **Archivo**, seleccione **Nuevo** y, a continuación, haga clic en **Proyecto** para abrir el cuadro de diálogo **Nuevo Proyecto**. Seleccione la plantilla de proyecto **Biblioteca de clases** en la lista de **Proyectos de Visual J#** y escriba **CDemoLib** en el cuadro **Nombre**.

**Sugerencia** Especifique siempre el nombre de un nuevo proyecto cuando lo cree. Al hacerlo se establece el paquete raíz, el nombre de ensamblado y el de proyecto, y también se garantiza que el componente predeterminado esté en el paquete correcto.

2. En el Explorador de soluciones, haga clic en **CDemoLib** con el botón secundario del *mouse* (ratón) y elija **Propiedades** en el menú contextual. Observe que el cuadro **Paquete predeterminado** contiene **CDemoLib**.

El paquete predeterminado se utiliza para calificar los nombres de los componentes del ensamblado. Por ejemplo, si dos ensamblados proporcionan componentes denominados `CDemo`, puede especificar el componente `CDemo` utilizando `CDemoLib.CDemo`.

Haga clic en **Cancelar** para cerrar el cuadro de diálogo.

3. En el menú **Proyecto**, elija **Agregar componente**.
4. En el cuadro de diálogo **Agregar nuevo elemento**, seleccione **Clase de componentes** y escriba **CDemo.jsl** en el cuadro **Nombre**. Haga clic en **Abrir**.

Se agregará un componente denominado `CDemo.jsl` a la biblioteca de clases.

5. En el Explorador de soluciones, haga clic con el botón secundario en **CDemo.jsl** y elija **Ver código**. Se abrirá el Editor de código.

Observe el código `extends System.ComponentModel.Component` inmediatamente después de `public class CDemo`. Esto designa la clase desde la que se hereda nuestra clase. De forma predeterminada, un componente hereda de la clase **Component** proporcionada por el sistema. La clase **Component** proporciona muchas funciones al componente, entre ellas la capacidad de utilizar diseñadores.

6. En el Explorador de soluciones, haga clic en **Class1.jsl** con el botón secundario del ratón y elija **Eliminar**. Así eliminará la clase predeterminada que se proporciona con la biblioteca de clases, ya que no se utilizará en este tutorial.
7. Desde el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Agregar constructores y un método Finalize

Los constructores controlan el modo en que se inicializa el componente; el método **Finalize** controla la forma en que se destruye. El código del constructor y del método **Finalize** de la clase `CDemo` mantiene una cuenta actualizada del número de objetos **CDemo** existentes.

### Para agregar código para el constructor y el método Finalize de la clase CDemo

1. En el Editor de código, agregue variables miembro al constructor **CDemo** para mantener un total actualizado de las instancias de la clase **CDemo** y un número de identificador para cada instancia.

```
// Visual J#
private long InstanceID;
private static long NextInstanceID = 0;
private static long ClassInstanceCount = 0;
```

Dado que las variables miembro `InstanceCount` y `NextInstanceID` se declaran **static**, sólo existen en el nivel de clase. Todas

las instancias de **CDemo** que tengan acceso a estos miembros utilizarán las mismas ubicaciones de memoria. Los miembros compartidos se inicializarán la primera vez que se haga referencia a la clase **CDemo** en el código. Puede ser la primera vez que se crea un objeto **CDemo** o la primera vez que se tiene acceso a uno de los miembros compartidos de la clase.

- Busque **public CDemo()** y **public CDemo(System.ComponentModel.IContainer container)**, los constructores predeterminados para la clase **CDemo**. En Visual J#, todos los constructores tienen el mismo nombre que la clase. El componente puede tener varios constructores, con diferentes parámetros, pero todos deben tener el mismo nombre que el propio componente.

**Nota** El nivel de acceso de los constructores de una clase determina qué clientes podrán crear instancias de la clase.

- Agregue el código siguiente a `public CDemo()`, para incrementar la cuenta de instancias al crear un nuevo objeto de la clase **CDemo** y para establecer el número de identificador de instancia.

**Nota** Agregue siempre el código después de la llamada a **InitializeComponent**. En este momento, todos los componentes constituyentes se habrán inicializado.

```
// Visual J#
InstanceID = NextInstanceID ++;
ClassInstanceCount ++;
```

**Nota** Los usuarios que conozcan la programación multiproceso observarán que la asignación `InstanceID` y el incremento de `NextInstanceID` debe ser una operación atómica.

- Agregue el método siguiente después del final del constructor:

```
// Visual J#
protected void Finalize()
{
    ClassInstanceCount --;
}
```

El administrador de memoria llama a **Finalize** inmediatamente antes de reclamar la memoria ocupada por el objeto **CDemo**. El método **Finalize** se origina en **Object**, la raíz de todos los tipos de referencia de .NET Framework. Si reemplaza **Finalize**, puede ejecutar la limpieza inmediatamente antes de quitar el componente de la memoria. No obstante, como verá más adelante en este tutorial, existen buenas razones para liberar antes los recursos.

## Agregar un método a la clase

La clase **CDemo** tiene un miembro que permite que el cliente averigüe cuántos objetos **CDemo** hay en la memoria en un momento dado.

### Para crear un método que recupere el número de instancias de CDemo

- Agregue la siguiente declaración de método a la clase **CDemo**, para permitir que los clientes recuperen el número de instancias de **CDemo**.

```
// Visual J#
public static long getInstanceCount()
{
    return ClassInstanceCount;
}
```

### Para generar el componente

- En el menú **Generar**, haga clic en **Generar**.

La generación debe ser correcta, sin errores ni advertencias del compilador.

## Probar el componente



Para probar el componente, necesitará un proyecto que lo utilice. Este proyecto debe ser el primero que se inicie al presionar el botón Ejecutar.

### Para agregar el proyecto de cliente **CDemoTest** como proyecto inicial para la solución

1. En el menú **Archivo**, seleccione **Agregar proyecto** y luego haga clic en **Nuevo proyecto** para abrir el cuadro de diálogo **Agregar nuevo proyecto**.
2. Seleccione la plantilla de proyecto **Aplicación para Windows** en la lista de proyectos de Visual J#, escriba **CDemoTest** en el cuadro **Nombre** y, a continuación, haga clic en **Aceptar**.
3. En el Explorador de soluciones, haga clic en **CDemoTest** con el botón secundario del *mouse* (ratón) y elija **Establecer como proyecto de inicio** en el menú contextual.

Para utilizar el componente **CDemo**, el proyecto cliente de prueba debe incluir una referencia al proyecto de la biblioteca de clases. Después de agregar la referencia, es una buena idea agregar una instrucción **import** a la aplicación de prueba, para simplificar el uso del componente.

### Para agregar una referencia al proyecto de biblioteca de clases

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo **References**, justo debajo de **CDemoTest**, y elija **Agregar referencia** en el menú contextual.
2. En el cuadro de diálogo **Agregar referencia**, seleccione la ficha **Proyectos**.
3. Haga doble clic en el proyecto de biblioteca de clases **CDemoLib**, para agregarlo a la lista **Componentes seleccionados**. **CDemoLib** aparecerá bajo el nodo **References** del proyecto **CDemoTest**. Haga clic en **Aceptar** para aceptar la selección.
4. En el Explorador de soluciones, haga clic en **Form1.jsl** con el botón secundario del *mouse* (ratón) y elija **Ver código** en el menú contextual.

Al agregar la referencia a **CDemoLib** podrá utilizar el nombre completo del componente **CDemo**, es decir **CDemoLib.CDemo**.

### Para agregar una instrucción **import**

- Agregue la siguiente instrucción **import** a la lista de instrucciones **import** que se encuentra en la parte superior del Editor de código para **Form1**.

```
// Visual J#  
import CDemoLib.*;
```

Si agrega la instrucción **import**, podrá omitir el nombre de biblioteca y hacer referencia al tipo de componente como **CDemo**.

Ahora creará y utilizará un programa de prueba para probar el componente.

## Tiempo de vida del objeto

El programa **CDemoTest** ilustrará el tiempo de vida del objeto en .NET Framework mediante la creación y liberación de grandes cantidades de objetos **CDemo**.

### Para agregar código de creación y liberación de objetos **CDemo**

1. Haga clic en **Form1.jsl[Diseño]** para volver al diseñador.
2. Arrastre un control **Button** y un control **Timer** desde la ficha **Windows Forms** del Cuadro de herramientas hasta la superficie de diseño de **Form1**.

El componente **Timer**, no visual, aparece en una superficie de diseño separada, debajo del formulario.

3. Haga doble clic en el icono de **timer1** para crear un método de control de eventos para el evento **Tick** del componente **Timer1**. Inserte el código siguiente en el método de control de eventos.

```
// Visual J#  
this.set_Text("CDemo instances: " + CDemo.GetInstanceCount());
```

A cada paso del temporizador, el título del formulario mostrará la cuenta actual de instancias de la clase **CDemo**. El nombre de clase se utiliza como calificador para la propiedad compartida **InstanceCount**; no es necesario crear una instancia de **CDemo** para tener acceso a un miembro compartido.

4. Busque el constructor de **Form1** (**public Form1()**) y agregue el código siguiente después de la llamada a

## InitializeComponent().

```
// Visual J#  
timer1.set_Enabled(true);
```

Esto iniciará el temporizador tan pronto como se cree el formulario.

5. Haga clic en la ficha **Form1.jsl [Diseño]** para volver al diseñador.
6. Haga doble clic en el control **Button** de Form1, para crear un método de control de eventos para el evento **Click** del botón. Inserte el código siguiente en el método de control de eventos.

```
// Visual J#  
CDemo cd;  
int ct;  
for (ct = 0; ct < 1000; ct++)  
{  
    cd = new CDemo();  
}
```

Es posible que este código le parezca extraño. Al crearse cada instancia de **CDemo**, la instancia anterior se libera. Cuando termine el bucle **for**, sólo quedará una instancia de **CDemo**. Al salir del método de control de eventos, incluso esta instancia será liberada, puesto que la variable `cd` quedará fuera de su ámbito.

Como ya habrá supuesto, las cosas no suceden de este modo.

## Para ejecutar y depurar los proyectos CDemoTest y CDemo

1. Presione F5 para iniciar la solución.

Se iniciará el proyecto de cliente y se mostrará el formulario Form1. Observe que el título del formulario muestra "CDemo instances: 0".

2. Haga clic en el botón. El título del formulario debe mostrar "CDemo instances: 1000".

Todas las instancias de **CDemo** se liberaron en el momento en que terminó el procedimiento de control del evento **Click** del botón. ¿Porqué no finalizaron? En pocas palabras, el administrador de memoria finaliza los objetos en segundo plano, con baja prioridad. La prioridad sólo aumenta si empieza a escasear la memoria del sistema. Este tipo de esquema de recolección de elementos no utilizados permite asignar objetos con mucha rapidez.

3. Haga clic en el botón varias veces más y observe el título. En algún momento, el número de instancias se reducirá repentinamente. Esto significa que el administrador de memoria reclamó la memoria de parte de los objetos.

**Nota** Si hace clic más de 10 veces y el número de instancias de **CDemo** no se reduce, es posible que necesite ajustar el código para que use más memoria. Cierre el formulario para volver al entorno de desarrollo y aumente el número de iteraciones del bucle **for** a 10000. A continuación, ejecute de nuevo el proyecto.

4. Repita el paso 3. Esta vez deberá esperar más antes de que el administrador de memoria finalice más objetos.

De hecho, cada vez que repita el paso 3, podrá asignar, probablemente, más objetos **CDemo** antes de que intervenga el administrador de memoria. Esto se debe a que se intercambia cada vez más memoria de Visual Studio, dejando más espacio para las instancias de **CDemo**.

5. Cierre el formulario para volver al entorno de programación.

## Vea también

[Programar con componentes](#) | [Tutoriales sobre la creación de componentes y controles](#)

# Tutorial: crear un componente sencillo multiproceso con Visual J#

Puede escribir aplicaciones que lleven a cabo varias tareas simultáneamente. Esta capacidad, denominada subprocesamiento múltiple o subprocesamiento libre, es un modo eficaz de diseñar componentes que hagan un uso intensivo del procesador y requieran acción del usuario. Un ejemplo de componente que haga uso del subprocesamiento múltiple sería uno que calcule información de nóminas. Este componente podría procesar los datos que un usuario incluyó en una base de datos en un subproceso mientras, en otro, se llevan a cabo cálculos de nómina con un uso intensivo del procesador. Al ejecutar estos procesos en subprocesos separados, los usuarios no tienen que esperar a que el equipo finalice los cálculos antes de especificar datos adicionales. En este tutorial, se creará un componente multiproceso simple que realiza varios cálculos complejos de forma simultánea.

## Crear el proyecto

La aplicación constará de un único formulario y un componente. El usuario especificará valores y hará una señal al componente para que comience los cálculos. El formulario recibirá entonces los valores del componente y los mostrará en controles Label. El componente realizará los cálculos con un uso intensivo del procesador y le hará una señal al formulario cuando haya terminado. Creará variables públicas en el componente para albergar los valores recibidos de la interfaz de usuario. También implementará métodos en el componente para realizar los cálculos en función de los valores de estas variables.

**Nota** Aunque suele ser preferible una función para un método que calcula un valor, no se pueden pasar argumentos entre subprocesos ni se pueden devolver valores. Hay varias maneras sencillas de suministrar valores a subprocesos y recibir valores de ellos. En esta demostración, devolverá valores a la interfaz de usuario actualizando variables públicas y se utilizarán eventos para enviar una notificación al programa principal cuando un subproceso haya terminado su ejecución.

## Para crear el formulario

1. Cree un proyecto nuevo de **aplicación para Windows**.
2. Denomine la aplicación **Calculations** y cambie el nombre de **Form1.jsl** a **frmCalculations.jsl**.

Este formulario servirá como interfaz de usuario principal para la aplicación.

3. En el Explorador de soluciones, haga clic con el botón secundario en **frmCalculations.jsl** y elija **Ver código**. Se abrirá el Editor de código.
4. En el menú **Edición**, elija **Buscar y reemplazar** y, a continuación, elija **Reemplazar**. Utilice **Reemplazar todo** para reemplazar **Form1** por **frmCalculations**.
5. En el Explorador de soluciones, haga clic con el botón secundario en **frmCalculations.jsl** y elija **Diseñador de vistas**. Se abrirá el diseñador.
6. Agregue al formulario cinco controles **Label**, cuatro controles **Button** y un control **TextBox**.
7. Establezca las propiedades de estos controles de la manera siguiente:

Control	Nombre	Texto
<b>Label1</b>	<b>lblFactorial1</b>	(En blanco)
<b>Label2</b>	<b>lblFactorial2</b>	(En blanco)
<b>Label3</b>	<b>lblAddTwo</b>	(En blanco)
<b>Label4</b>	<b>lblRunLoops</b>	(En blanco)
<b>Label5</b>	<b>lblTotalCalculations</b>	(En blanco)
<b>Button1</b>	<b>btnFactorial1</b>	<b>Factorial</b>
<b>Button2</b>	<b>btnFactorial2</b>	<b>Factorial - 1</b>
<b>Button3</b>	<b>btnAddTwo</b>	<b>Add Two</b>
<b>Button4</b>	<b>btnRunLoops</b>	<b>Run a Loop</b>
<b>Textbox1</b>	<b>txtValue</b>	(En blanco)

## Para crear el componente Calculator

1. En el menú **Proyecto**, elija **Agregar componente**.
2. Asigne al componente el nombre **Calculator**.

## Para agregar variables públicas al componente Calculator

1. Abra el Editor de código para **Calculator**.
2. Agregue instrucciones para crear las variables públicas que utilizará para pasar valores de **frmCalculations** a cada subproceso.

La variable `varTotalCalculations` mantendrá un total actualizado del número de cálculos realizados por el componente, y otras variables recibirán los valores del formulario.

```
// Visual J#
public int varAddTwo;
public int varFact1;
public int varFact2;
public int varLoopValue;
public double varTotalCalculations = 0;
```

### Para agregar métodos y eventos al componente Calculator

1. Declare los delegados para los eventos que utilizará el componente para comunicar valores al formulario.

**Nota** Aunque va a declarar cuatro eventos, sólo necesitará crear tres delegados, porque dos eventos tendrán la misma firma.

Inmediatamente después de las declaraciones de variable especificadas en el paso anterior, agregue el siguiente código:

```
// Visual J#
// This delegate will be invoked with two of your events.
/**@delegate */
public delegate void FactorialCompleteHandler(double Factorial, double TotalCalculations)
;
/**@delegate */
public delegate void AddTwoCompleteHandler(int Result, double TotalCalculations);
/**@delegate */
public delegate void LoopCompleteHandler(double TotalCalculations, int Counter);
```

2. Declare los eventos que utilizará el componente para comunicarse con la aplicación agregando el siguiente código inmediatamente después del código incluido en el paso anterior.

```
// Visual J#
public FactorialCompleteHandler FactorialComplete;
public FactorialCompleteHandler FactorialMinusOneComplete;
public AddTwoCompleteHandler AddTwoComplete;
public LoopCompleteHandler LoopComplete;

/** @event */
public void add_FactorialComplete(FactorialCompleteHandler e)
{
    FactorialComplete = (FactorialCompleteHandler) System.Delegate.Combine(this.FactorialComplete, e);
}
/** @event */
public void remove_FactorialComplete(FactorialCompleteHandler e)
{
    FactorialComplete = (FactorialCompleteHandler) System.Delegate.Remove(this.FactorialComplete, e);
}

/** @event */
public void add_FactorialMinusOneComplete(FactorialCompleteHandler e)
{
    FactorialMinusOneComplete = (FactorialCompleteHandler) System.Delegate.Combine(this.FactorialComplete, e);
}
```

```

        FactorialMinusOneComplete, e);
    }
    /** @event */
    public void remove_FactorialMinusOneComplete(FactorialCompleteHandler e)
    {
        FactorialMinusOneComplete = (FactorialCompleteHandler) System.Delegate.Remove(this.Fac
        torialMinusOneComplete, e);
    }

    /** @event */
    public void add_AddTwoComplete(AddTwoCompleteHandler e)
    {
        AddTwoComplete = (AddTwoCompleteHandler) System.Delegate.Combine(this.AddTwoComplete,
        e);
    }
    /** @event */
    public void remove_AddTwoComplete(AddTwoCompleteHandler e)
    {
        AddTwoComplete = (AddTwoCompleteHandler) System.Delegate.Remove(this.AddTwoComplete, e
        );
    }

    /** @event */
    public void add_LoopComplete(LoopCompleteHandler e)
    {
        LoopComplete = (LoopCompleteHandler) System.Delegate.Combine(this.LoopComplete, e);
    }
    /** @event */
    public void remove_LoopComplete(LoopCompleteHandler e)
    {
        LoopComplete = (LoopCompleteHandler) System.Delegate.Remove(this.LoopComplete, e);
    }
}

```

3. Inmediatamente después del código que agregó en el paso anterior, agregue el siguiente código:

```

// Visual J#
// This method will calculate the value of a number minus 1 factorial
// (varFact2-1!).
public void FactorialMinusOne()
{
    double varTotalAsOfNow = 0;
    double varResult = 1;
    // Performs a factorial calculation on varFact2 - 1.
    for (int varX = 1; varX <= varFact2 - 1; varX++)
    {
        varResult *= varX;
        // Increments varTotalCalculations and keeps track of the
        // current total as of this instant.

        varTotalCalculations += 1;
        varTotalAsOfNow = varTotalCalculations;

    }
    // Signals that the method has completed and communicates the
    // result and a value of total calculations performed up to this
    // point.
    FactorialMinusOneComplete.Invoke(varResult, varTotalAsOfNow);
}

```

```

// This method will calculate the value of a number factorial.
// (varFact1!)
public void Factorial()
{
    double varResult = 1;
    double varTotalAsOfNow = 0;
    for (int varX = 1; varX <= varFact1; varX++)
    {
        varResult =varResult * varX;

        varTotalCalculations =varTotalCalculations + 1;
        varTotalAsOfNow = varTotalCalculations;

    }
    //FactorialComplete(varResult, varTotalAsOfNow);
    FactorialComplete.Invoke( varResult, varTotalAsOfNow );
}

// This method will add 2 to a number (varAddTwo+2).
public void AddTwo()
{
    double varTotalAsOfNow = 0;
    int varResult = varAddTwo + 2;

    varTotalCalculations = varTotalCalculations + 1;
    varTotalAsOfNow = varTotalCalculations;

    AddTwoComplete.Invoke(varResult, varTotalAsOfNow);
}

// This method will run a loop with a nested loop varLoopValue times.
public void RunALoop()
{
    int varX;
    double varTotalAsOfNow = 0;
    for (varX = 1; varX <= varLoopValue; varX++)
    {
        // This nested loop is added to slow down the program
        // and create a processor-intensive application.
        for (int varY = 1; varY <= 500; varY++)
        {
            varTotalCalculations += 1;
            varTotalAsOfNow = varTotalCalculations;
        }
    }
    LoopComplete.Invoke(varTotalAsOfNow, varLoopValue);
}

```

## Transferir datos proporcionados por el usuario al componente

El paso siguiente consiste en agregar código a **frmCalculations** para recibir datos proporcionados por el usuario y para transferir y recibir valores a y desde el componente Calculator.

### Para implementar funcionalidad cliente a frmCalculations

1. Abra **frmCalculations** en el Editor de código.
2. Busque la instrucción `public class frmCalculations`. Inmediatamente después de `{`, agregue:

```
Calculator Calculator1;
```

```
// Visual J#
```

3. Busque el constructor. Inmediatamente después de `InitializeComponent()`, agregue las líneas siguientes:

```
// Visual J#
// Creates a new instance of Calculator.
Calculator1 = new Calculator();
```

4. En el diseñador, haga clic en cada botón para generar el esquema de código de los controladores de eventos Click de cada control y agregue código para crear los controladores.

Cuando termine, los controladores de eventos Click deben tener el siguiente aspecto:

```
// Visual J#
private void btnFactorial1_Click (Object sender, System.EventArgs e)
{
    Calculator1.varFact1 = Integer.parseInt(txtValue.get_Text());
    // Disables the btnFactorial1 until this calculation is complete.
    btnFactorial1.set_Enabled(false);
    Calculator1.Factorial();
}

private void btnFactorial2_Click (Object sender, System.EventArgs e)
{
    Calculator1.varFact2 = Integer.parseInt(txtValue.get_Text());
    btnFactorial2.set_Enabled(false);
    Calculator1.FactorialMinusOne();
}

private void btnAddTwo_Click (Object sender, System.EventArgs e)
{
    Calculator1.varAddTwo = Integer.parseInt(txtValue.get_Text());
    btnAddTwo.set_Enabled(false);
    Calculator1.AddTwo();
}

private void btnRunLoops_Click (Object sender, System.EventArgs e)
{
    Calculator1.varLoopValue = Integer.parseInt(txtValue.get_Text());
    btnRunLoops.set_Enabled(false);
    // Let the user know that a loop is running.
    lblRunLoops.set_Text( "Looping");
    Calculator1.RunALoop();
}
```

5. Después del código que agregó en el paso anterior, agregue el siguiente código para controlar los eventos que recibirá el formulario desde **Calculator1**:

```
// Visual J#
protected void FactorialHandler(double Value, double Calculations)
// Displays the returned value in the appropriate label.
{
    lblFactorial1.set_Text(""+ Value);
    // Re-enables the button so it can be used again.
    btnFactorial1.set_Enabled( true);
    // Updates the label that displays the total calculations
    // performed.
    lblTotalCalculations.set_Text("TotalCalculations are " +
        Calculations);
}
```

```

protected void FactorialMinusHandler(double Value, double Calculations)
{
    lblFactorial2.set_Text(""+Value);
    btnFactorial2.set_Enabled(true);
    lblTotalCalculations.set_Text("TotalCalculations are " + Calculations);
}

protected void AddTwoHandler(int Value, double Calculations)
{
    lblAddTwo.set_Text(""+Value);
    btnAddTwo.set_Enabled(true);
    lblTotalCalculations.set_Text("TotalCalculations are " + Calculations);
}

protected void LoopDoneHandler(double Calculations, int Count)
{
    btnRunLoops.set_Enabled(true);
    lblRunLoops.set_Text(""+Count);
    lblTotalCalculations.set_Text("TotalCalculations are " + Calculations);
}

```

6. En el constructor de **frmCalculations**, agregue el siguiente código inmediatamente después de la línea `Calculator1 = new Calculator();` para controlar los eventos personalizados que recibirá el formulario desde **Calculator1**.

```

// Visual J#
Calculator1.add_FactorialComplete(new Calculator.FactorialCompleteHandler(FactorialHandle
r));
Calculator1.add_FactorialMinusOneComplete(new Calculator.FactorialCompleteHandler(this.Fa
ctorialMinusHandler));
Calculator1.add_AddTwoComplete(new Calculator.AddTwoCompleteHandler(this.AddTwoHandler));
Calculator1.add_LoopComplete(new Calculator.LoopCompleteHandler(this.LoopDoneHandler));

```

## Probar la aplicación

Ha creado un proyecto que incorpora un formulario y un componente capaz de realizar varios cálculos complejos. Aunque no ha implementado aún la capacidad de subprocesamiento múltiple, probará el proyecto para comprobar su funcionalidad antes de continuar.

### Para probar el proyecto

1. Presione F5 para iniciar la solución.

La aplicación se inicia y aparece **frmCalculations**.

2. En el cuadro de texto, escriba **4** y haga clic en el botón **Add Two**.

La etiqueta que aparece debajo debe mostrar el número "6" y "Total Calculations are 1.0" debe aparecer en **lblTotalCalculations**.

3. Ahora haga clic en el botón **Factorial - 1**.

Debe aparecer el número "6.0" debajo del botón y ahora **lblTotalCalculations** muestra "Total Calculations are 4.0".

4. Cambie el valor del cuadro de texto a **20** y haga clic en el botón **Factorial**.

Debe aparecer el número "2.43290200817664E18" debajo y **lblTotalCalculations** muestra ahora "Total Calculations are 24.0".

5. Cambie el valor del cuadro de texto a **50000** y haga clic en el botón **Run a Loop**.

Observe que hay un pequeño pero perceptible intervalo antes de que el botón vuelva a estar habilitado. La etiqueta que aparece debajo del botón debe mostrar "50000" y el total de cálculos debe ser "2.5000024E7".



6. Cambie el valor del cuadro de texto a **500000** y haga clic en el botón **Run a Loop**; inmediatamente después haga clic en el botón **Add Two** dos veces.

El botón no responde y tampoco ningún control del formulario hasta que terminan los bucles. Cierre el programa.

Si el programa ejecuta sólo un subproceso de ejecución, los cálculos de uso intensivo del procesador como los del ejemplo anterior tienden a paralizar el programa hasta que terminan los cálculos. En la sección siguiente, agregará capacidad de subprocesamiento múltiple a la aplicación para que se puedan ejecutar varios subprocesos a la vez.

## Agregar capacidad de subprocesamiento múltiple

El ejemplo anterior mostraba las limitaciones de las aplicaciones que ejecutan un único subproceso. En la sección siguiente, utilizará el objeto de clase [Thread](#) para agregar varios subprocesos de ejecución al componente.

### Para agregar el método **Threads**

1. Abra **Calculator.jsl** en el Editor de código.
2. Agregue las siguientes variables miembro a la clase **Calculator**:

```
// Visual J#
public System.Threading.Thread FactorialThread;
public System.Threading.Thread FactorialMinusOneThread;
public System.Threading.Thread AddTwoThread;
public System.Threading.Thread LoopThread;
```

3. Inmediatamente antes del final de la declaración de clase al final del archivo, agregue el método siguiente:

```
// Visual J#
public void ChooseThreads(int threadNumber)
{
    // Determines which thread to start based on the value it receives.
    switch(threadNumber)
    {
        case 1:
            // Sets the thread using the AddressOf the method where
            // the thread will start.
            FactorialThread = new System.Threading.Thread(new System.Threading.ThreadStar
t(this.Factorial));
            // Starts the thread.
            FactorialThread.Start();
            break;
        case 2:
            FactorialMinusOneThread = new System.Threading.Thread(new System.Threading.Th
readStart(this.FactorialMinusOne));
            FactorialMinusOneThread.Start();
            break;
        case 3:
            AddTwoThread = new System.Threading.Thread(new
System.Threading.ThreadStart(this.AddTwo));
            AddTwoThread.Start();
            break;
        case 4:
            LoopThread = new System.Threading.Thread(new
System.Threading.ThreadStart(this.RunALoop));
            LoopThread.Start();
            break;
    }
}
```

Cuando se crea una instancia de un objeto **Thread**, ésta requiere un argumento en la forma de un objeto **ThreadStart**. El objeto **ThreadStart** es un delegado que apunta a la dirección del método donde debe comenzar el subproceso. Un objeto

**ThreadStart** no puede aceptar parámetros ni pasar valores y, por tanto, sólo puede indicar un método **void**. El método **ChooseThreads** que implementó recibirá un valor del programa que lo llama y utilizará ese valor para determinar el subproceso que debe iniciar.

### Para agregar el código correspondiente a frmCalculations

1. Abra el archivo **frmCalculations.jsl** en el Editor de código y busque `private void btnFactorial1_Click`.
  - a. Convierta en comentario la línea que llama al método **Calculator1.Factorial1** directamente como se muestra a continuación:

```
// Visual J#  
// Calculator1.Factorial()
```

- b. Agregue la línea siguiente para llamar al método **Calculator1.ChooseThreads**:

```
// Visual J#  
// Passes the value 1 to Calculator1, thus directing it to start the  
// correct thread.  
Calculator1.ChooseThreads(1);
```

2. Realice cambios similares para los demás métodos **button\_click**.

**Nota** Asegúrese de incluir el valor correcto para el argumento **Threads**.

Cuando termine, el código debe ser similar al siguiente:

```
// Visual J#  
private void btnFactorial1_Click (Object sender, System.EventArgs e)  
{  
    Calculator1.varFact1 = Integer.parseInt(txtValue.get_Text());  
    // Disables the btnFactorial1 until this calculation is complete.  
    btnFactorial1.set_Enabled(false);  
    //Calculator1.Factorial();  
    Calculator1.ChooseThreads(1);  
}  
  
private void btnFactorial2_Click (Object sender, System.EventArgs e)  
{  
    Calculator1.varFact2 = Integer.parseInt(txtValue.get_Text());  
    btnFactorial2.set_Enabled(false);  
    //Calculator1.FactorialMinusOne();  
    Calculator1.ChooseThreads(2);  
}  
  
private void btnAddTwo_Click (Object sender, System.EventArgs e)  
{  
    Calculator1.varAddTwo = Integer.parseInt(txtValue.get_Text());  
    btnAddTwo.set_Enabled(false);  
    //Calculator1.AddTwo();  
    Calculator1.ChooseThreads(3);  
}  
  
private void btnRunLoops_Click (Object sender, System.EventArgs e)  
{  
    Calculator1.varLoopValue = Integer.parseInt(txtValue.get_Text());  
    btnRunLoops.set_Enabled(false);  
    // Let the user know that a loop is running.  
    lblRunLoops.set_Text( "Looping");  
    //Calculator1.RunALoop();  
    Calculator1.ChooseThreads(4);  
}
```

```
}
```

## Llamadas de cálculo de referencias a controles

Ahora permitirá actualizar la vista del formulario. Puesto que el subproceso de ejecución principal posee siempre controles, cualquier llamada a un control desde un subproceso subordinado requiere una llamada de cálculo de referencias. El cálculo de referencias es el acto de pasar una llamada a través de límites de subprocesos y consume muchos recursos. Para minimizar la cantidad de cálculo de referencias necesario y estar seguro de que las llamadas se controlan de un modo seguro para subprocesos, utilizará el método [Control.BeginInvoke](#) para invocar a los métodos del subproceso de ejecución principal. Este tipo de llamada es necesario cuando se realizan llamadas a métodos que procesan controles. Para obtener información detallada, vea [Manipular controles de subprocesos](#).

### Para crear los procedimientos que invocan a controles

1. Abra el Editor de código para **frmCalculations**. En la sección de declaraciones, agregue el código siguiente:

```
// Visual J#  
/**@delegate */  
public delegate void FHandler(double Value, double Calculations);  
/**@delegate */  
public delegate void A2Handler(int Value, double Calculations);  
/**@delegate */  
public delegate void LDHandler(double Calculations, int Count);
```

**Invoke** y **BeginInvoke** requieren un delegado para el método correspondiente como un argumento. Estas líneas declaran las firmas de delegado que utilizará **BeginInvoke** para invocar los métodos correspondientes.

2. Agregue los siguientes métodos vacíos al código:

```
// Visual J#  
public void FactHandler(double Value, double Calculations)  
{  
}  
public void Fact1Handler(double Value, double Calculations)  
{  
}  
public void Add2Handler(int Value, double Calculations)  
{  
}  
public void LDoneHandler(double Calculations, int Count)  
{  
}
```

3. En el menú **Edición**, utilice **Cortar** y **Pegar** para cortar todo el código del método `FactorialHandler` y pegarlo en `FactHandler`.
4. Repita el paso anterior de `FactorialMinusHandler` a `Fact1Handler`, de `AddTwoHandler` a `Add2Handler` y de `LoopDoneHandler` a `LDoneHandler`.

Cuando termine, no debe quedar código en `FactorialHandler`, `Factorial1Handler`, `AddTwoHandler` y `LoopDoneHandler`, y todo el código que contenían deben haberse movido a los correspondientes métodos nuevos.

5. Llame al método **BeginInvoke** para invocar a los métodos de manera asíncrona. Puede llamar a **BeginInvoke** desde el formulario (**this**) o desde cualquiera de sus controles.

Una vez terminado, el código debe ser similar al siguiente:

```
// Visual J#  
protected void FactorialHandler(double Value, double Calculations)  
// Displays the returned value in the appropriate label.  
{  
    this.BeginInvoke(new FHandler(FactHandler), new Object[] {(System.Double)Value, (Syst
```

```

em.Double)Calculations});
}

protected void FactorialMinusHandler(double Value, double Calculations)
{
    this.BeginInvoke(new FHandler(Fact1Handler), new Object [] {(System.Double)Value, (System.Double)Calculations});
}

protected void AddTwoHandler(int Value, double Calculations)
{
    this.BeginInvoke(new A2Handler(Add2Handler), new Object[] {(System.Int32)Value, (System.Double) Calculations});
}

protected void LoopDoneHandler(double Calculations, int Count)
{
    this.BeginInvoke(new LDHandler(LDoneHandler), new Object[] {(System.Double)Calculations, (System.Int32)Count});
}

```

Puede parecer que el controlador de eventos está haciendo sólo una llamada al método siguiente. Sin embargo, da lugar a que se invoque un método en el subproceso principal. Este sistema ahorra llamadas fuera de los límites de los subprocesos y permite a las aplicaciones multiproceso ejecutarse de forma eficiente y sin peligro de que se produzcan bloqueos. Para obtener información detallada, vea [Manipular controles de subprocesos](#).

6. Guarde su trabajo.

7. Pruebe la solución eligiendo **Inicio** en el menú **Depurar**.

a. Escriba **1000000** en el cuadro de texto y haga clic en **Run a loop**.

Aparecerá "Looping" en la etiqueta situada debajo del botón. Este bucle debe tardar un tiempo considerable en ejecutarse. Si termina demasiado pronto, ajuste el tamaño del número en consecuencia.

b. En una sucesión rápida, haga clic en los tres botones que están habilitados. Verá que todos los botones responden a su acción. La etiqueta situada debajo de **Add Two** debe ser la primera en mostrar un resultado. Los resultados se mostrarán en las etiquetas que aparecen debajo de los botones de factorial. Estos resultados evalúan con infinito, puesto que el número devuelto por un factorial de 1.000.000 es demasiado grande para que lo contenga una variable de precisión doble. Finalmente, tras una demora adicional, los resultados se devuelven debajo del botón **Run a Loop**.

Como acaba de observar, se llevaron a cabo cuatro conjuntos de cálculos distintos de manera simultánea en cuatro subprocesos separados. La interfaz de usuario permanece lista para responder a la acción del usuario y los resultados se devuelven después de terminar cada subproceso.

## Coordinar los subprocesos

Un usuario con experiencia en aplicaciones multiproceso puede percibir un sutil defecto con el código tal y como está escrito. Llame de nuevo a las líneas de código desde los métodos de cálculo de **Calculator.jsl**:

```

// Visual J#
varTotalCalculations = varTotalCalculations + 1;
varTotalAsOfNow = varTotalCalculations;

```

Estas dos líneas de código aumentan la variable pública **varTotalCalculations** y definen la variable local **varTotalAsOfNow** con este valor. Dicho valor se devuelve a **frmCalculations** y se muestra en un control de etiqueta. Pero no se sabe si el valor es correcto. Si sólo se está ejecutando un subproceso, la respuesta claramente es sí. Pero si se están ejecutando varios subprocesos, la respuesta se vuelve incierta. Cada subproceso tiene la capacidad de incrementar la variable **varTotalCalculations**. Es posible que, después de que un subproceso incremente esta variable, pero antes de que copie el valor en **varTotalAsOfNow**, otro subproceso altere el valor de la variable incrementándolo. Esto da lugar a la posibilidad de que, en realidad, cada subproceso informe de resultados imprecisos.

Utilice la palabra clave **synchronized** para permitir la sincronización de subprocesos con el fin de garantizar que cada subproceso devuelva siempre un resultado exacto. La sintaxis es la siguiente:

```
// Visual J#
synchronized(AnObject)
{
    // Insert code that affects the object.
    // Insert more code that affects the object.
    // Insert more code that affects the object.
    // Release the lock.
}
```

Cuando se incluye el bloque **synchronized**, la ejecución en la expresión especificada se bloquea hasta que el subproceso especificado tiene un bloqueo exclusivo en el objeto en cuestión. En el ejemplo anterior, la ejecución está bloqueada en `AnObject`. La ejecución puede proceder entonces como un bloque sin interferencias de otros subprocesos. Un conjunto de instrucciones que se ejecutan como una unidad se denomina atómico. Cuando se encuentra el signo `}`, se libera la expresión y se permite que los subprocesos continúen con normalidad.

### Para agregar el bloque sincronizado a la aplicación

1. Abra **Calculator.jsl** en el Editor de código.
2. Busque cada instancia del siguiente código:

```
// Visual J#
varTotalCalculations = varTotalCalculations + 1;
varTotalAsOfNow = varTotalCalculations;
```

Debe haber cuatro instancias de este código, una en cada método de cálculo.

3. Modifique el código de cada instancia para que tenga el siguiente aspecto:

```
// Visual J#
synchronized(this)
{
    varTotalCalculations = varTotalCalculations + 1;
    varTotalAsOfNow = varTotalCalculations;
}
```

4. Guarde el trabajo y pruébelo como en el ejemplo anterior.

Es posible que note que el rendimiento del programa se ve afectado. Esto se debe a que la ejecución de los subprocesos se detiene cuando se obtiene un bloque exclusivo en el componente. Aunque garantiza la precisión, esta solución impide algunas de las ventajas de rendimiento de varios subprocesos. Debe considerar detenidamente la necesidad de bloquear subprocesos e implementarlos sólo cuando sea absolutamente necesario.

### Vea también

[Programar con componentes](#) | [Subprocesamiento múltiple en componentes](#) | [Coordinar varios subprocesos de ejecución](#) | [Tutoriales sobre la creación de componentes y controles](#)

# Tutorial: crear un control de usuario con Visual J#

Los controles de usuario proporcionan un medio para crear y reutilizar interfaces gráficas personalizadas. Un control de usuario es básicamente un componente con una representación visual. Como tal, puede constar de uno o más componentes, bloques de código o controles de formularios Windows Forms. Estos elementos pueden ampliar la funcionalidad validando la acción del usuario, modificando las propiedades de vista o llevando a cabo otras tareas requeridas por el autor. Los controles de usuario se pueden colocar en formularios Windows Forms del mismo modo que otros controles. En la primera parte del tutorial, creará un control de usuario sencillo denominado **ctlClock**. En la segunda parte del tutorial, ampliará la funcionalidad de **ctlClock** a través de herencia.

## Crear el proyecto

Cuando se crea un proyecto nuevo, se especifica el nombre para definir el paquete raíz, el nombre de ensamblado y el nombre de proyecto, así como garantizar que el componente predeterminado estará en el paquete correcto.

### Para crear la biblioteca de controles **ctlClockLib** y el control **ctlClock**

1. En el menú **Archivo**, seleccione **Nuevo** y, a continuación, haga clic en **Proyecto** para abrir el cuadro de diálogo **Nuevo Proyecto**.
2. Seleccione la plantilla de proyecto **Biblioteca de controles de Windows** en la lista de **Proyectos de Visual J#** y escriba **ctlClockLib** en el cuadro **Nombre**.

El nombre de proyecto, **ctlClockLib**, se asigna también al paquete raíz de forma predeterminada. El paquete raíz se utiliza para calificar los nombres de los componentes del ensamblado. Por ejemplo, si dos ensamblados proporcionan componentes denominados `ctlClock`, puede especificar el componente `ctlClock` utilizando `ctlClockLib.ctlClock`.

3. En el Explorador de soluciones, haga clic en **UserControl1** con el botón secundario del *mouse* (ratón) y elija **Ver código** en el menú contextual.
4. Busque la instrucción **class**, `public class UserControl1`, y cambie **UserControl1** a **ctlClock** para cambiar el nombre del componente.

**Nota** De forma predeterminada, un control de usuario se hereda de la clase **UserControl** proporcionada por el sistema. Esta clase proporciona la funcionalidad que requieren todos los controles de usuario e implementa métodos y propiedades estándar.

5. Busque el constructor, `public UserControl1()` y cambie `UserControl1` a `ctlClock`.
6. En el Explorador de soluciones, haga clic en **UserControl1** y, a continuación, en la ventana **Propiedades**, cambie la propiedad **FileName** a **ctlClock.jsl**.
7. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Agregar controles y componentes de Windows al control de usuario

Una interfaz visual es una parte esencial del control de usuario. Esta interfaz se implementa agregando uno o más controles al diseñador del control de usuario. En la siguiente demostración, incorporará controles al control de usuario y escribirá código para implementar la funcionalidad.

### Para agregar un control **Label** y un control **Timer** al control de usuario

1. En el Explorador de soluciones, haga clic con el botón secundario en **ctlClock.jsl** y seleccione **Diseñador de vistas**.
2. En el Cuadro de herramientas, haga clic en la ficha **Windows Forms** y haga doble clic en **Label**.

Se agregará un control **Label** denominado **label1** al control en el **Diseñador del control de usuario**.

3. En el diseñador, haga clic en **label1**. En la ventana **Propiedades**, defina las propiedades siguientes:

Propiedad	Cambiar a
<b>Name</b>	<b>lblDisplay</b>
<b>Text</b>	<b>(espacio en blanco)</b>
<b>TextAlign</b>	<b>MiddleCenter</b>
<b>Font.Size</b>	<b>14</b>

4. En la barra de herramientas, haga clic en **Windows Forms** y, a continuación, haga doble clic en **Timer**.

Puesto que **timer** es un componente, no tiene representación visual en tiempo de ejecución. Por tanto, no aparece con los

controles en el **Diseñador del control de usuario**, sino en la bandeja de componentes.

5. En la bandeja de componentes, haga clic en **timer1** y defina la propiedad **Interval** con el valor **1000**, y la propiedad **Enabled** como **true**.

La propiedad **Interval** controla la frecuencia con la que el componente timer marca los pasos. Cada vez que **timer1** marca un paso, ejecuta el código del evento **timer1\_Tick**. El intervalo representa el número de milisegundos entre pasos.

6. En la bandeja de componentes, haga doble clic en **timer1** para ir al evento **timer1\_Tick** de **ctlClock**.
7. Modifique el código para que tenga el siguiente aspecto:

```
// Visual J#
protected void timer1_Tick (Object sender, System.EventArgs e)
{
    lblDisplay.set_Text(System.DateTime.get_Now().ToLongTimeString());
}
```

Este código dará lugar a que se muestre la hora actual en **lblDisplay**. Puesto que el intervalo de **timer1** se definió como 1000, este evento se activará cada mil milisegundos, actualizando así la hora actual cada segundo.

De manera predeterminada, el método **timer1\_Tick** agregado haciendo doble clic a **timer1** tendrá ámbito privado. Cambie el ámbito a protegido.

8. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Agregar propiedades personalizadas al control de usuario

El control de reloj encapsula ahora un control **Label** y un componente **Timer**, cada uno con su propio conjunto de propiedades heredadas. Si bien las propiedades individuales de estos controles no serán accesibles para posteriores usuarios del control, puede crear y exponer propiedades personalizadas escribiendo los bloques de código pertinentes. En la siguiente sección, agregará propiedades personalizadas al control que permitan al usuario cambiar el color del fondo y del texto.

### Para agregar un miembro personalizado al control de usuario

1. En el Explorador de soluciones, haga clic en **ctlClock.jsl** con el botón secundario del *mouse* (ratón) y después haga clic en **Ver código** en el menú contextual.

Se abrirá el Editor de código.

2. Busque la instrucción `public class ctlClock`. Después de la llave de apertura `{`, escriba:

```
// Visual J#
private Color colFColor;
private Color colBColor;
```

Estas instrucciones crean las variables privadas que utilizará para almacenar los valores de las propiedades personalizadas que va a crear.

3. Agregue el código siguiente después de las declaraciones de variable del paso 2:

```
// Visual J#
/** @property */
public Color get_ClockBackColor()
{
    return colBColor;
}
/** @property */
public void set_ClockBackColor(Color value)
{
    colBColor = value;
    lblDisplay.set_BackColor(colBColor);
}
/** @property */
public Color get_ClockForeColor()
```

```

{
    return colFColor;
}
/** @property */
public void set_ClockForeColor(Color value)
{
    colFColor = value;
    lblDisplay.set_ForeColor(colFColor);
}

```

El código anterior crea dos propiedades personalizadas, **ClockForeColor** y **ClockBackColor**, disponibles para posteriores usuarios de este control. Las instrucciones **get\_** y **set\_** permiten almacenar y recuperar los valores de las propiedades personalizadas, así como código para implementar la funcionalidad correspondiente.

4. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Probar el control

Los controles no son aplicaciones independientes; deben estar alojados en un contenedor. Para probar el control, debe proporcionar un proyecto de prueba dentro del cual se ejecuta. En esta sección, generará el control y lo probará en un formulario Windows Forms.

### Para generar el control

- En el menú **Generar**, haga clic en **Generar**.

### Para crear un proyecto de prueba

1. En el menú **Archivo**, seleccione **Agregar proyecto** y haga clic en **Nuevo proyecto** para abrir la ventana **Agregar nuevo proyecto**.
2. Haga clic en **Aplicación para Windows** y, en el cuadro **Nombre**, escriba **Test**.
3. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo **References** del proyecto de prueba. Haga clic en **Agregar referencia** para mostrar el cuadro de diálogo del mismo nombre.
4. Haga clic en la ficha **Proyectos**. El proyecto del control de usuario aparecerá en la lista **Nombre de proyecto**.
5. Haga doble clic en el proyecto.

Ahora, el proyecto aparecerá en la ventana **Componentes seleccionados**.

Una vez que haya agregado una referencia, podrá colocar el control en el formulario.

### Para probar el control

1. En el Cuadro de herramientas, haga clic en **Mis controles de usuario** y, a continuación, desplácese hasta que se muestre el icono de control que representa **ctlClock**.
2. Haga doble clic en este icono.

Esta acción agregará una copia del control al formulario. Observe que muestra la hora actual y se actualiza cada segundo.

3. Seleccione el icono y desplace el *mouse* (ratón) sobre el formulario.
4. Mantenga presionado el botón primario del *mouse* mientras lo mueve por el formulario.

Esta acción incluye otra copia del control en el formulario. Puede agregar tantas copias del temporizador como desee al formulario.

5. En el diseñador, haga clic en una de las instancias de **ctlClock**.

Aparecerán las propiedades de esta instancia en la ventana **Propiedades**.

6. En esta ventana, busque la propiedad **ClockBackColor** y selecciónela para mostrar la paleta de colores.
7. Elija un color haciendo clic en él.

El color de fondo del control cambiará al color seleccionado.

8. Utilice una secuencia de eventos similar para comprobar que la propiedad **ClockForeColor** funciona como se esperaba.

En esta sección, ha visto cómo se pueden combinar componentes y controles de Windows con código y empaquetamiento para



proporcionar funcionalidad personalizada en forma de control de usuario. Ha aprendido a exponer propiedades en un control de usuario y probarlo una vez terminado. En la siguiente sección, aprenderá a construir un control de usuario heredado utilizando **ctlClock** como base.

## Heredar de un control de usuario

En la sección anterior, aprendió a combinar controles de Windows, componentes y código en controles de usuario reutilizables. Ahora puede utilizar un control de usuario como base sobre la que construir otros controles. El proceso de derivación de una clase a partir de una clase base se denomina herencia. En esta sección, creará un control de usuario denominado **ctlAlarmClock**. Este control se derivará de su control principal, **ctlClock**. Aprenderá a ampliar la funcionalidad de **ctlClock** reemplazando los métodos principales y agregando métodos y propiedades nuevos.

## Crear el control heredado

El primer paso para crear un control heredado es derivarlo de su principal. Esta acción crea un control nuevo que tiene todas las propiedades, los métodos y las características gráficas del control principal, pero también puede actuar como base para la adición de funcionalidad nueva o modificada.

### Para crear el control heredado

1. En el Explorador de soluciones, haga clic en **ctlClockLib**.
2. En el menú **Proyecto**, elija **Agregar control heredado**.

Se abrirá la ventana **Agregar nuevo elemento**, con la opción **Control de usuario heredado** seleccionada.

3. En el cuadro **Nombre**, escriba **ctlAlarmClock.jsl** y haga clic en **Abrir**.

Aparecerá la ventana **Selector de herencia**.

4. En **Nombre de componente**, haga doble clic en **ctlClock**.
5. En el Explorador de soluciones, examine los proyectos actuales. Se ha agregado un archivo denominado **ctlAlarmClock.jsl**.

## Agregar propiedades de alarma

A un control heredado se le agregan propiedades del mismo modo que se agregan a un control de usuario. Ahora utilizará la sintaxis de declaración de propiedades para agregar dos propiedades al control:

- **AlarmTime**, que almacenará el valor de fecha y hora en que sonará la alarma.
- **AlarmSet**, que indicará si la alarma está puesta o no.

### Para agregar propiedades al control de usuario

1. En el Explorador de soluciones, haga clic con el botón secundario en **ctlAlarmClock** y elija **Ver código**.
2. Busque la instrucción `public class`. Observe que el control se hereda de **ctlClockLib.ctlClock**. Debajo de la instrucción **public class ctlAlarmClock extends ctlClockLib.ctlClock** {}, agregue el siguiente código:

```
// Visual J#
private System.DateTime dteAlarmTime;
private boolean blnAlarmSet;
// These properties will be declared as public to allow future
// developers to access them.
/** @property*/
public System.DateTime get_AlarmTime()
{
    return dteAlarmTime;
}
/** @property*/
public void set_AlarmTime(System.DateTime value)
{
    dteAlarmTime = value;
}
/** @property*/
public boolean get_AlarmSet()
{
```

```

        return blnAlarmSet;
    }
    /** @property*/
    public void set_AlarmSet(boolean value)
    {
        blnAlarmSet = value;
    }

```

## Agregar a la interfaz gráfica del control

El control heredado tiene una interfaz visual que es idéntica al control del que se hereda. Posee los mismos controles constituyentes que el control principal, pero las propiedades de estos controles no estarán disponibles a menos que se expongan de manera específica. Puede agregar a la interfaz gráfica de un control de usuario heredado del mismo modo que agregaría a cualquier control de usuario. Para continuar agregando a la interfaz visual de su reloj de alarma, agregará un control Label que parpadeará cuando esté sonando la alarma.

### Para agregar el control Label

1. En el Explorador de soluciones, haga clic en **ctlAlarmClock** con el botón secundario del *mouse* (ratón) y elija **Diseñador de vistas** en el menú contextual.

Aparecerá el diseñador de **ctlAlarmClock** en la ventana principal.

2. Haga clic en la parte de la vista del control y observe la ventana Propiedades.

Observe que, mientras se muestran las propiedades, están atenuadas. Esto indica que estas propiedades son nativas de **IblDisplay** y no se pueden modificar ni se puede tener acceso a ellas en la ventana Propiedades. De manera predeterminada, los controles contenidos en un control de usuario son privados (**private**) y no se puede tener acceso a sus propiedades de ningún modo.

**Sugerencia** Si desea que posteriores usuarios del control tengan acceso a sus controles internos, declárelos como públicos (**public**) o protegidos (**protected**). Esto permitirá definir y modificar propiedades de los controles contenidos en el control de usuario utilizando el código apropiado.

3. Agregue un control **Label** al control de usuario.
4. Con el *mouse* (ratón), mueva el control Label justo debajo del cuadro de vista. En la ventana **Propiedades**, defina las propiedades siguientes:

Propiedad	Valor
Name	IblAlarm
Text	Alarm!
TextAlign	Middle Center
Visible	False

## Agregar la funcionalidad de alarma

En las secciones anteriores, agregó miembros personalizados y un control que proporcionará funcionalidad de alarma al control de usuario. En esta sección, agregará código para comparar la hora actual con la hora de la alarma y, si coinciden, iniciará la alarma. Reemplazará el método **timer1\_Tick** de **ctlClock** para ampliar la capacidad de **ctlAlarmClock** mientras conserva toda la funcionalidad heredada de **ctlClock**.

### Para reemplazar el método timer1\_Tick de ctlClock

1. En el Editor de código, busque la instrucción `private boolean blnAlarmSet;`. Justo después, agregue la siguiente instrucción:

```

// Visual J#
private boolean blnColorTicker;

```

2. Agregue el código siguiente justo antes del final de la declaración de la clase:

```

// Visual J#
protected void timer1_Tick(Object sender, System.EventArgs e)

```

```

{
    // Calls the timer1_Tick method of ctlClock.
    super.timer1_Tick(sender, e);
    // Checks to see if the alarm is set.
    if (blnAlarmSet == false)
        return;
    else
        // If the date, hour, and minute of the alarm time are the same as
        // now, cause the display to flash.
        {
            if (dteAlarmTime.get_Date() == System.DateTime.get_Now().get_Date() &&
                dteAlarmTime.get_Hour() == System.DateTime.get_Now().get_Hour() &&
                dteAlarmTime.get_Minute() == System.DateTime.get_Now().get_Minute())
            {
                // Makes lblAlarm visible, and changes the backcolor based
                // on the value of blnColorTicker. The backcolor of the
                // label will flash once per tick of the clock.
                lblAlarm.set_Visible(true);
                if (blnColorTicker == false)
                {
                    lblAlarm.set_BackColor( Color.get_Red());
                    blnColorTicker = true;
                }
                else
                {
                    lblAlarm.set_BackColor(Color.get_Blue());
                    blnColorTicker = false;
                }
            }
            else
            {
                // Once the alarm has sounded for a minute, the label is
                // made invisible again.
                lblAlarm.set_Visible(false);
            }
        }
    }
}

```

La adición de este código realiza varias tareas. Indica al control que utilice este método en lugar del método que se heredó del control base. Cuando se llama a este método, invoca a la instrucción `super.timer1_Tick`, garantizando así que se reproduce en este control toda la funcionalidad incorporada en el control original. A continuación, ejecuta código adicional para incorporar la funcionalidad de alarma. Aparecerá un control Label parpadeante cuando se desencadene la alarma.

El control de reloj está casi listo. Sólo falta implementar un modo de desactivarlo. Para ello, agregaremos un botón y código al método **btnAlarmOff\_Click**.

#### Para implementar el método shutoff

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en **ctlAlarmClock.jsl** y elija **Diseñador de vistas**.

Se abrirá el diseñador.

2. Agregue un botón al control. Defina las propiedades del botón como sigue:

Propiedad	Valor
Name	btnAlarmOff
Text	Disable Alarm

3. En el diseñador, haga doble clic en **btnAlarmOff**.

Se abrirá el Editor de código en la línea `private void btnAlarmOff_Click`.

4. Modifique este método para que tenga el siguiente aspecto:

```
// Visual J#
private void btnAlarmOff_Click (Object sender, System.EventArgs e)
{
    // Turns off the alarm.
    blnAlarmSet = false;
    // Hides the flashing label
    lblAlarm.setVisible(false);
}
```

5. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Probar el control heredado

De igual modo que con un control de usuario estándar, un control de usuario heredado no puede ser independiente, sino que debe estar alojado en un formulario u otro contenedor. Puesto que **ctlAlarmClock** tiene más funcionalidad, es necesario código adicional para probarlo. En esta sección, escribirá un programa sencillo para probar la funcionalidad de **ctlAlarmClock**. Escribirá código para definir y mostrar la propiedad **AlarmTime** de **ctlAlarmClock** y probar sus funciones heredadas.

### Para generar y agregar el control a un formulario de prueba

1. En el Explorador de soluciones, haga clic en **ctlClockLib**. En el menú **Generar**, elija **Generar ctlClockLib**.
2. Agregue un nuevo proyecto de **Aplicación para Windows** a la solución y asígnele el nombre **Test2**.
3. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo **References** del proyecto de prueba. Haga clic en **Agregar referencia** para mostrar la ventana del mismo nombre. Haga clic en la ficha **Proyectos**. Aparecerá **ctlClockLib** en la lista **Nombre de proyecto**. Haga doble clic en él y observe que ahora aparece en la ventana **Componentes seleccionados**. Haga clic en Aceptar para agregarlo como referencia.
4. En el Cuadro de herramientas, haga clic en la ficha formularios **Mis controles de usuario**.
5. Busque el icono **ctlAlarmClock**.
6. Haga doble clic en él para agregar una copia de **ctlAlarmClock** al formulario.
7. En el cuadro de herramientas, haga doble clic en **DateTimePicker** para agregar un control **DateTimePicker** al formulario, y agregue un control **Label** haciendo doble clic en **Label**.
8. Utilice el *mouse* (ratón) para colocar los controles en un lugar apropiado del formulario.
9. Defina las propiedades de estos controles de la manera siguiente:

Control	Propiedad	Valor
label1	Text	(Déjela en blanco)
	Name	lblTest
dateTimePicker1	Name	dtpTest
	Format	Time

10. En el diseñador, haga doble clic en **dtpTest**.

Se abrirá el Editor de código en `private void dtpTest_ValueChanged`.

11. Modifique el código para que tenga el siguiente aspecto:

```
// Visual J#
private void dtpTest_ValueChanged (Object sender, System.EventArgs e)
{
    ctlAlarmClock1.set_AlarmTime(dtpTest.get_Value());
    ctlAlarmClock1.set_AlarmSet(true);
    lblTest.set_Text("Alarm Time is " + ctlAlarmClock1.get_AlarmTime().ToShortTimeString(
));
}
```

12. En el Explorador de soluciones, haga clic en **Test2** con el botón secundario del *mouse* (ratón) y elija **Establecer como proyecto de inicio** en el menú contextual.
13. En el menú **Depurar**, elija **Iniciar**.

Se iniciará el programa de prueba. Observe que la hora actual se actualiza en el control **ctlAlarmClock** y que la hora de

inicio se muestra en el control **DateTimePicker**.

14. Haga clic en **DateTimePicker** donde se muestran los minutos.
15. Con el uso del teclado, defina un valor para los minutos que sea un minuto más que la hora actual que muestra **ctlAlarmClock**.

La hora de la configuración de la alarma se muestra en **lblTest**.

16. Espere que la hora mostrada coincida con la hora de la alarma.

Cuando la hora mostrada coincida con la hora a la que está puesta la alarma, se iniciará **lblAlarm**. Desactive la alarma haciendo clic en el botón **Disable Alarm**. Ahora puede restablecer la alarma.

Este tutorial ha cubierto una serie de conceptos clave. Ha aprendido a crear un control de usuario combinando controles y componentes en un contenedor de control de usuario. Ha aprendido a agregar propiedades al control y escribir código para implementar funcionalidad personalizada. En la segunda sección, aprendió a ampliar la funcionalidad de un control de usuario dado a través de herencia, y modificar la funcionalidad de métodos host reemplazándolos.

## Vea también

[Programar con componentes](#) | [Tutorial: heredar de un formulario Windows Forms con Visual J#](#) | [Tutoriales sobre la creación de componentes y controles](#)

# Tutorial: crear una clase de colección propia con Visual J#

Puede crear clases de colección propias mediante la herencia de una de las muchas clases de colección de .NET Framework y la adición de código que implemente la funcionalidad personalizada propia. En este tema, utilizará la herencia para crear una colección sencilla, y con fuerte control de tipos, heredada de **CollectionBase**.

El entorno .NET Framework proporciona varias clases de tipo colección en el espacio de nombres **System.Collections**. Algunas, tales como **Stack**, **Queue** y **Dictionary**, son clases especializadas que se implementaron para desempeñar funciones específicas. Otras, tales como **CollectionBase** y **DictionaryBase**, son clases **MustInherit** que tienen ya cierta funcionalidad básica, pero que dejan para el programador la mayor parte de la implementación.

La clase **CollectionBase** tiene ya implementaciones para el método **Clear** y la propiedad **Count**; mantiene una propiedad **Protected** denominada **List**, que utiliza para la organización y el almacenamiento internos. Otros métodos, tales como **Add** y **Remove**, y la propiedad **Item**, requieren implementación. Aplicará la propiedad **Item** implementando el método **get\_Item** del descriptor de acceso **get**.

En este tutorial, utilizará la clase **CollectionBase** para crear una clase denominada **WidgetCollection**. Se trata de una colección que sólo acepta widgets y expone sus miembros como tipos **Widget**, en lugar de aceptar objetos y exponer miembros como tipo **Object**. A continuación, implementará métodos para agregar widgets a la colección y quitar el widget correspondiente a un índice específico; también implementará un método **get\_Item** que devuelve el objeto widget del índice correspondiente. El primer paso es crear una clase **Widget** para insertarla en **WidgetCollection**.

## Para crear la clase **Widget**

1. Abra una nueva aplicación para Windows y asígnele el nombre **WidgetProject**. En el menú **Proyecto**, elija **Agregar clase**. En el cuadro de diálogo **Agregar nuevo elemento**, dé el nombre **Widget.jsl** a la clase.

Se abrirá el Editor de código para la clase **Widget**.

2. Agregue una variable `Name` pública a la clase **Widget**. El código debe ser similar al siguiente:

```
// Visual J#
public class Widget
{
    public String Name;
    public Widget()
    {
    }
}
```

3. En el menú **Archivo**, elija **Guardar todo**.

Acaba de crear una clase **Widget** para utilizarla con la clase **WidgetCollection** que creará a continuación. La clase **Widget** es una clase sencilla que sirve para ilustrar cómo funcionará la colección con establecimiento inflexible de tipos. El siguiente paso es crear la clase **WidgetCollection**.

## Para crear la clase **WidgetCollection**

1. En el menú **Proyecto**, elija **Agregar clase**. En el cuadro de diálogo **Agregar nuevo elemento**, dé el nombre **WidgetCollection.jsl** a la clase.

Se abrirá el Editor de código para la clase **WidgetCollection**.

2. En el editor, agregue código para hacer que la clase se herede de **CollectionBase**, como en este ejemplo:

```
// Visual J#
public class WidgetCollection extends System.Collections.CollectionBase
{
}
}
```

3. En el menú **Archivo**, elija **Guardar todo**.

Habrá creado la clase **WidgetCollection**. Dado que esta clase se hereda de **CollectionBase**, ya tiene gran parte de la funcionalidad de la colección implementada, como se muestra a continuación.

- El método **Clear** borra la colección.
- La propiedad **Count** mantiene un seguimiento del número de miembros actuales.
- Un objeto **Protected**, denominado **List**, que se puede utilizar para hacer un seguimiento de los widgets.

Sin embargo, no hay ninguna propiedad **Add**, **Remove** o **Item** (que se implementará como un método denominado **get\_Item** en Visual J#). Su implementación se deja al programador. Ahora, implementará el método **Add** de modo que **WidgetCollection** sólo pueda recibir objetos **Widget**.

### Para implementar el método Add

1. Agregue el siguiente código a la clase **WidgetCollection**, debajo de la instrucción **class**:

```
// Visual J#
// Indicates that only widgets can be added to the collection.
public void Add(Widget aWidget)
{
    get_List().Add(aWidget);
}
```

En este método, los elementos que se pueden agregar al objeto **List** a través del argumento del método **Add** se restringen al tipo **Widget**. Aunque el objeto **List** puede aceptar cualquier tipo de objeto, este método prohíbe que se agreguen objetos de cualquier tipo, excepto **Widget**, y actúa como envoltorio del objeto **List**.

2. En el menú **Archivo**, elija **Guardar todo**.

Ahora dispone de un medio para agregar widgets a la colección. A continuación, debe implementar un medio para quitarlos. Hará esto de forma similar a como creó el método **Add**, mediante la creación de un método **Remove** que admita como argumento un índice y, a su vez, llame al método **List.RemoveAt**.

### Para implementar el método Remove

1. Bajo el método **Add**, agregue el código siguiente:

```
// Visual J#
public void Remove(int index)
{
    // Check to see if there is a widget at the supplied index.
    if (index > (this.get_Count() - 1) || index < 0)
    // If no widget exists, a message box is shown and the operation
    // is cancelled.
    {
        System.Windows.Forms.MessageBox.Show("Index not valid!");
    }
    else
    {
        get_List().RemoveAt(index);
    }
}
```

Este método acepta un valor entero para el argumento **index**. Si el valor es válido, éste se pasa al método **RemoveAt** del objeto **List**; como consecuencia, el elemento que se encuentra en el índice indicado se quita de la colección.

2. En el menú **Archivo**, elija **Guardar todo**.

Deberá implementar un último elemento para completar la funcionalidad básica de la colección: se trata del método **get\_Item**. El método **get\_Item** permite obtener una referencia a un objeto de la colección, mediante una referencia al índice. Puesto que ya dispone de un método **Add** para agregar miembros a la colección, la propiedad **Item** será de sólo lectura en esta demostración; sin embargo, no tiene por qué serlo en otros contextos.

### Para implementar el método get\_Item

1. Bajo el método **Add**, agregue el código siguiente:

```
/**
```

```

    * @property
    */
    public Object get_Item(int Index)
    {
        // The appropriate item is retrieved from the List object and
        // explicitly cast to the Widget type, then returned to the
        // caller.
        return (Widget) get_List().get_Item(Index);
    }

```

2. En el menú **Archivo**, elija **Guardar todo**.

Habrás implementado la funcionalidad de colección básica en la clase. Implementó métodos para agregar y quitar widgets de la colección y acaba de implementar una propiedad que devuelve una referencia widget al elemento correspondiente. En la próxima sección, probará la colección **WidgetCollection**.

## Generar el proyecto de prueba

Ahora que la clase **WidgetCollection** está completa, es el momento de probar su funcionalidad. Para ello, creará una aplicación para Windows sencilla que agregará, quitará y se desplazará en bucle por los widgets de la colección.

### Para crear el proyecto de prueba

1. En el **Explorador de soluciones**, haga clic con el botón secundario en **Form1** y elija **Diseñador de vistas**.

Se abrirá el diseñador para Form1.

2. Mediante el cuadro de herramientas, agregue dos controles **Label**, dos controles **TextBox** y tres controles **Button** a Form1.
3. Establezca las propiedades de la manera siguiente:

Control	Propiedad	Valor
label1	Text	Widget Name
label2	Text	Widget Index
textBox1	Text	Déjela en blanco
textBox2	Text	Déjela en blanco
button1	Text	Add a Widget
button2	Text	Remove a Widget
button3	Text	Review Widgets

En el diseño del formulario, **label1** debe ser la etiqueta de **textBox1**, y **label2** debe ser la etiqueta de **textBox2**.

4. En el Diseñador de formularios, haga clic con el botón secundario en Form1 y elija **Ver código**.

Se abrirá el Editor de código para Form1.

5. En el Editor de código, busque el inicio de la declaración de clase Form1. Justo después, agregue el siguiente código:

```

// Visual J#
WidgetCollection myWidgetCollection = new WidgetCollection();

```

6. En el diseñador, haga doble clic en **button1**.

El Editor de código se centra en el controlador de eventos `button1_Click`.

7. En el controlador de eventos `button1_Click`, agregue el código del nuevo widget. Defina el nombre de este widget con el valor `textBox1`. El código debe ser similar al siguiente:

```

// Visual J#
if (textBox1.get_Text().CompareTo("")==0)
    MessageBox.Show("Please name your widget.");
else
{
    // Declares and instantiates a new widget.
    Widget aWidget = new Widget();
}

```



```

aWidget.Name = textBox1.get_Text();
// Adds the new widget to the collection.
myWidgetCollection.Add(aWidget);
// Updates textbox2 with the index of the widget that was added.
// The index of a zero-based collection is Count property minus 1.
textBox2.set_Text("" + (myWidgetCollection.get_Count() - 1));
}

```

8. Agregue un controlador de eventos a button2. En el método **button2\_Click**, agregue código para quitar el widget en el índice especificado en **textBox2**. A continuación se muestra un ejemplo de este código:

```

// Visual J#
myWidgetCollection.Remove(Integer.parseInt(textBox2.get_Text()));

```

No hay necesidad de comprobar si existe un valor válido para el índice en **textBox2**, ya que el método **WidgetCollection.Remove** lo comprueba automáticamente.

9. Agregue un controlador de eventos a **button3**. En el método **button3\_Click**, escriba código para recorrer todos los miembros de la colección y mostrar secuencialmente sus nombres en un cuadro de mensaje. A continuación se muestra una forma de hacerlo:

```

// Visual J#
int counter;
for (counter = 0; counter <= myWidgetCollection.get_Count() - 1; counter++)
{
    MessageBox.Show(((Widget)myWidgetCollection.get_Item(counter)).Name);
}

```

10. En el Explorador de soluciones, haga clic en con el botón secundario del *mouse* en **WidgetProject** y elija **Propiedades**. Se abrirá la página de propiedades de WidgetProject.
11. En **Objeto inicial**, elija **WidgetProject.Form1** en el menú y haga clic en **Aceptar** para cerrar la página.

## Probar la colección

Ahora ejecutará la aplicación de prueba. Probará el método **Add**, el método **Remove** y la recuperación de objetos Widget de la colección.

### Para probar la colección

1. Presione F5 para iniciar la aplicación.
2. En el cuadro de texto **Widget Name**, escriba **Widget0** y haga clic en el botón **Add a Widget**. Repita esto dos veces, sustituyendo **Widget0** por **Widget1** y **Widget2**.  
  
Se habrán agregado tres widgets a la colección.
3. Haga clic en el botón **Review Widgets**.  
  
En un cuadro de mensaje aparecerán sucesivamente los nombres de cada uno de los widgets.
4. En el cuadro de texto **Índice de Widget**, escriba **1** y haga clic en el botón **Quitar un Widget**.  
  
Se quitará el widget de índice 1. El índice se actualiza dinámicamente, así que el widget que previamente ocupaba el índice 2 ocupa ahora el índice 1.
5. Haga clic en el botón **Review Widgets**.  
  
En un cuadro de mensaje aparecerán sucesivamente, de nuevo, los nombres de cada uno de los widgets. Observe que ya no se muestra "Widget1", puesto que se quitó de la colección.

En este tutorial aprendió a implementar la funcionalidad básica de una colección mediante la herencia de la clase **System.Collections.CollectionBase**. Aprendió a crear una colección con establecimiento inflexible de tipos y a implementar los métodos **Add** y **Remove**, así como la propiedad **Item** (método **get\_Item** en Visual J#). Para obtener más información acerca de las clases base para la creación de clases de colección, vea [System.Collections \(Espacio de nombres\)](#).

## Vea también

[System.Collections \(Espacio de nombres\)](#) | [Administración de objetos con colecciones](#) | [Utilizar colecciones en clases](#) | [Tutoriales sobre la creación de componentes y controles](#)

# Tutorial: heredar de un formulario Windows Forms con Visual J#

Visual J# proporciona la capacidad de crear eficaces controles personalizados utilizando herencia. La herencia permite crear controles que mantienen toda la funcionalidad heredada de controles de formularios Windows Forms estándar, pero también incorporan funcionalidad personalizada. En este tutorial, creará un sencillo control heredado denominado **ValueButton**. Este botón heredará la funcionalidad del botón de formularios Windows Forms estándar, y expondrá un miembro personalizado denominado **ButtonValue**.

## Crear el proyecto

Cuando se crea un proyecto nuevo, se especifica el nombre para definir el paquete raíz, el nombre de ensamblado y el nombre de proyecto, así como para garantizar que el componente predeterminado estará en el paquete correcto.

### Para crear la biblioteca de clases ValueButtonLib y el control ValueButton

1. En el menú **Archivo**, seleccione **Nuevo** y, a continuación, haga clic en **Proyecto** para abrir el cuadro de diálogo **Nuevo Proyecto**.
2. Seleccione la plantilla de proyecto **Biblioteca de controles de Windows** en la lista de **Proyectos de Visual J#** y escriba **ValueButtonLib** en el cuadro **Nombre**.

El nombre de proyecto, **ValueButtonLib**, se asigna también al paquete raíz de manera predeterminada. El paquete raíz se utiliza para calificar los nombres de los componentes del ensamblado. Por ejemplo, si dos ensamblados proporcionan componentes denominados `ValueButton`, puede especificar el componente `ValueButton` utilizando `ValueButtonLib.ValueButton`. Para obtener más información, vea [Componentes y ensamblados](#).

3. En el Explorador de soluciones, haga clic con el botón secundario en **UserControl1.jsl** y elija **Ver código**.
4. Busque la instrucción **class**, `public class UserControl1`, y cambie **UserControl1** a **ValueButton** para cambiar el nombre del componente.
5. Busque el constructor, `public UserControl1()`, y cámbielo a `public ValueButton()`.
6. En la instrucción **class**, cambie el tipo del que hereda este control de **System.Windows.Forms.UserControl** a **System.Windows.Forms.Button**. Esto permite que el control heredado herede toda la funcionalidad del control **Button**.
7. En el Explorador de soluciones, haga clic en **UserControl1.jsl** y, a continuación, en la ventana **Propiedades**, cambie la propiedad **FileName** a **ValueButton.jsl**.
8. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

Observe que ya no hay disponible ningún diseñador. Puesto que el control **Button** crea su propia apariencia, no puede modificarla en el diseñador. Su representación visual será exactamente la misma que la de la clase de la que hereda (es decir, **Button**) a menos que se modifique en el código.

## Agregar un miembro al control heredado

Un uso posible de los controles de formularios Windows Forms heredados es la creación de controles que son idénticos a los controles de formularios Windows Forms estándar, pero que exponen propiedades personalizadas. En esta sección, agregará un miembro denominado **ButtonValue** al control.

### Para agregar el miembro ButtonValue

1. En el Explorador de soluciones, haga clic en **ValueButton.jsl** con el botón secundario del *mouse* (ratón) y, después, haga clic en **Ver código** en el menú contextual.
2. Busque la instrucción `class`. Inmediatamente después de `{`, agregue el siguiente código:

```
// Visual J#
// Creates the private variable that will store the value of your
// member.
private int varValue;
// Declares the methods to access or modify the member.
/** @property */
public int get_ButtonValue()
{
    return varValue;
}
```

```
}  
/** @property */  
public void set_ButtonValue(int value)  
{  
    varValue = value;  
}
```

Este código define los métodos por los que se almacena y recupera el miembro **ButtonValue**. La instrucción **get\_** define el valor devuelto con el valor almacenado en la variable privada **varValue**, y la instrucción **set\_** define el valor de la variable privada utilizando la palabra clave **value**.

3. En el menú **Archivo**, elija **Guardar todo** para guardar el proyecto.

## Probar el control

Los controles no son proyectos independientes; deben estar alojados en un contenedor. Para probar el control, debe proporcionar un proyecto de prueba en el que ejecutarlo. También debe hacer que el control sea accesible para el proyecto de prueba generándolo. En esta sección, generará el control y lo probará en un formulario Windows Forms.

### Para generar el control

- En el menú **Generar**, haga clic en **Generar**.

La generación debe ser correcta, sin errores ni advertencias del compilador.

### Para crear un proyecto de prueba

1. En el menú **Archivo**, seleccione **Agregar proyecto** y haga clic en **Nuevo proyecto** para abrir el cuadro de diálogo **Agregar nuevo proyecto**.
2. Seleccione el nodo **Proyectos de Visual J#** y haga clic en **Aplicación para Windows**.
3. En el cuadro **Nombre**, escriba **Test**.
4. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* (ratón) en el nodo **References** y seleccione **Agregar referencia** en el menú contextual para mostrar el cuadro de diálogo del mismo nombre.
5. Haga clic en la ficha **Proyectos**.
6. Haga doble clic en el proyecto **ValueButtonLib** y observe que ahora aparece en el panel **Componentes seleccionados**.

Después de agregar la referencia, agregue el nuevo control al Cuadro de herramientas. Si el control aparece ya en el Cuadro de herramientas, omita la sección siguiente.

7. En el Explorador de soluciones, haga clic con el botón secundario en **Test** y después en **Establecer como proyecto de inicio** en el menú contextual.

### Para agregar el control al Cuadro de herramientas

1. Haga clic con el botón secundario del *mouse* (ratón) en el Cuadro de herramientas y elija **Agregar o quitar elementos** en el menú contextual.

Aparecerá el cuadro de diálogo **Personalizar cuadro de herramientas**.

2. Elija la ficha **Componentes de .NET Framework** y haga clic en **Examinar**. En la carpeta **ValueButtonLib\bin\debug**, seleccione **ValueButtonLib.dll**.

Aparecerá **ValueButton** en la lista de componentes en el cuadro de diálogo **Personalizar cuadro de herramientas**.

3. En este cuadro de diálogo, active la casilla que aparece junto a **ValueButton** y cierre la ventana.

**ValueButton** se agrega a la ficha del Cuadro de herramientas seleccionada.

### Para agregar el control al formulario

1. En el Explorador de soluciones, haga clic con el botón secundario en **Form1.jsl** y elija **Diseñador de vistas** en el menú contextual.
2. En el cuadro de herramientas, busque el icono **ValueButton**. Haga doble clic en este icono.

Aparecerá una instancia de **ValueButton** en el formulario.

3. Haga clic con el botón secundario en **ValueButton** y seleccione **Propiedades** en el menú contextual.
4. En la ventana **Propiedades**, examine las propiedades de este control. Observe que son idénticas a las propiedades de un botón estándar.
5. Defina el miembro **ButtonValue** con el valor 5 agregando el siguiente código al constructor `public Form1` después de llamar a `InitializeComponent`.

```
// Visual J#  
valueButton1.set_ButtonValue(5);    // Sets ButtonValue member of inherited button to 5.
```

6. En la ficha **Windows Forms** del Cuadro de herramientas, haga doble clic en **Label** para agregar un control **Label** al formulario.
7. Coloque de nuevo la etiqueta en el centro del formulario.
8. Haga doble clic en **valueButton1**.

El Editor de código se abre en el evento **valueButton1\_Click**.

9. Agregue la línea de código siguiente:

```
// Visual J#  
label1.set_Text("" + valueButton1.get_ButtonValue());
```

10. En el Explorador de soluciones, haga clic en **Test** con el botón secundario del *mouse* (ratón) y elija **Establecer como proyecto de inicio** en el menú contextual.
11. En el menú **Depurar**, elija **Iniciar**.

Aparecerá **Form1**.

12. Haga clic en **valueButton1**.

Aparecerá el número '5' en Label1, mostrando que se ha pasado el miembro ButtonValue del control heredado a Label1 a través del método **valueButton1\_Click**. Por tanto, el control **ValueButton** hereda toda la funcionalidad del botón de Windows Forms estándar, pero expone un miembro adicional personalizado.

## Vea también

[Programar con componentes](#) | [Tutorial: crear un control de usuario con Visual J#](#) | [Tutoriales sobre la creación de componentes y controles](#)

# Tutoriales sobre el componente de servicios Framework

Temas paso a paso que muestran varios elementos integrados en Windows, tales como registros de eventos, contadores de rendimiento y servicios.

## En esta sección

### [Explorar registros de eventos, orígenes de eventos y entradas con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar registros de eventos y sus entradas, y escribir entradas adicionales en ellos.

### [Instalar un componente de registro de eventos con Visual J#](#)

Explica cómo se utiliza un componente de instalación para configurar recursos de servidor necesarios para la instancia del componente **EventLog**.

### [Crear una cola y trabajar con mensajes en Visual J#](#)

Explica cómo utilizar una instancia de componente **MessageQueue** para que interactúe con colas de mensajes de Windows.

### [Cambiar y recuperar valores de los contadores de rendimiento en Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar valores sin formato y valores calculados de un contador mediante el componente **PerformanceCounter**.

### [Recuperar categorías y contadores con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo recuperar listas de categorías de contadores de rendimiento, así como sobre los contadores que éstas contienen.

### [Administrar un proceso de Windows con Visual J#](#)

Presenta instrucciones paso a paso sobre cómo crear una instancia de un componente **Process** y utilizarlo para interactuar con un proceso de Windows.

### [Reaccionar a eventos del sistema de archivos con Visual J#](#)

Explica cómo utilizar una instancia de componente **FileSystemWatcher** para supervisar archivos y directorios, y reaccionar cuando se produzcan cambios.

### [Agregar objetos Active Directory con Visual J#](#)

Explica cómo utilizar el componente **DirectorySearcher** para interactuar con las estructuras del sistema de archivos de Active Directory.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, componentes y controles, trabajar con Windows Forms y Web Forms, y desarrollar tareas relacionadas con datos.

### [Tutoriales sobre la creación de componentes y controles](#)

Tutoriales que le orientarán en el uso de componentes, controles y subprocesamiento.

### [Programar con componentes](#)

Sección de Visual Basic/Visual C# que explica los componentes de todos los tipos, incluidos los utilizados en servidores.

# Tutorial: explorar registros de eventos, orígenes de eventos y entradas con Visual J#

Este tutorial le guiará a través de las principales áreas funcionales del registro de eventos en una aplicación de Visual Studio. Durante este tutorial aprenderá a:

- Crear un componente **EventLog**.
- Escribir código para crear y eliminar registros de eventos personalizados.
- Escribir entradas de varios tipos en el registro personalizado.
- Leer entradas del registro personalizado.
- Comprobar la existencia de registros y orígenes de eventos.
- Borrar entradas de registro.
- Utilizar el Explorador de servidores para comprobar los resultados de las acciones de registro de eventos.

## Crear la interfaz de usuario

En este tutorial, creará un formulario Windows Forms y utilizará una serie de controles para iniciar una serie de acciones de registro de eventos.

### Para crear el formulario y los controles para la aplicación

1. En el cuadro de diálogo **Nuevo proyecto**, cree una **aplicación de Visual J# para Windows** y asígnele el nombre **EventLogApp1**.
2. Agregue ocho botones al formulario Windows Forms y establezca las siguientes propiedades para ellos:

Control	Propiedad Text	Propiedad Name
Button1	Crear registro personalizado	CreateLog
Button2	Eliminar registro	DeleteLog
Button3	Escribir entrada	WriteEntry
Button4	Borrar registro	ClearLog
Button5	Comprobar que existe el registro	VerifyLog
Button6	Comprobar que existe el origen	VerifySource
Button7	Quitar origen de eventos	RemoveSource
Button8	Leer entrada	ReadEntry

3. A continuación, haga lo siguiente con cada botón:
  - a. En el diseñador, haga doble clic en el botón para crear un controlador de eventos predeterminado para el botón. Aparecerá el Editor de código y se abrirá un procedimiento stub para el evento **Click** del botón.
  - b. Vuelva a la vista Diseño y haga doble clic en el siguiente botón.
  - c. Continúe hasta haber creado un procedimiento stub de controlador de eventos predeterminado para cada uno de los botones.
4. En la ficha **Componentes** del Cuadro de herramientas, arrastre un componente **EventLog** hasta el formulario.

Aparecerá una instancia del componente **EventLog** en el área de bandeja de componentes, en la parte inferior del formulario.

## Crear y eliminar un registro personalizado

En este procedimiento, utilizará el método **SourceExists** para comprobar que el archivo de código fuente que va a utilizar no existe ya. Si existe, lo eliminará y llamará al método **CreateEventSource**. El sistema creará un registro personalizado cuando se ejecute este código.

### Para crear el registro personalizado

1. En el Editor de código, agregue una instrucción **import** al principio del archivo que haga referencia al espacio de nombres **System.Diagnostics**.

```
// Visual J#
import System.Diagnostics.*;
```

2. Busque el procedimiento **CreateLog\_Click**.
3. Escriba el código siguiente:

```
// Visual J#
if (!(EventLog.SourceExists("Source1")))
{
    EventLog.CreateEventSource("Source1", "NewLog1");
}
else
{
    EventLog.DeleteEventSource("Source1");
    EventLog.CreateEventSource("Source1", "NewLog1");
}
```

### Para eliminar un registro personalizado

1. En el Editor de código, busque el procedimiento **DeleteLog\_Click**.
2. Escriba el código siguiente:

```
// Visual J#
if (EventLog.SourceExists("Source1"))
{
    EventLog.Delete("NewLog1");
}
```

### Escribir entradas en el registro

En este procedimiento, utilizará la instancia del componente **EventLog** que creó para escribir entradas en el registro. Para hacerlo así, en primer lugar configurará el componente para que utilice la cadena de origen recién creada; a continuación, especificará dos entradas para escribirlas: un evento informativo y un evento de error.

#### Para escribir entradas en el registro

1. En el Editor de código, busque el procedimiento **WriteEntry\_Click**.
2. Establezca la propiedad **Source** de la instancia del componente **EventLog** en Source1, que es la cadena de origen que utilizó al crear el registro personalizado. Utilice este código:

```
// Visual J#
eventLog1.set_Source("Source1");
```

3. Llame al método **WriteEntry** y especifique un mensaje de texto. Utilice este código:

```
// Visual J#
eventLog1.WriteEntry("This is an informational message");
```

4. Llame otra vez al método **WriteEntry** y utilice el formulario sobrecargado para especificar tanto un mensaje como un tipo de error. Utilice este código:

```
// Visual J#
eventLog1.WriteEntry("This is an error message", EventLogEntryType.Error);
```

### Borrar entradas de registro

En este procedimiento, utilizará el método **Clear** para quitar entradas existentes del registro personalizado.

#### Para borrar entradas de registro

1. En el Editor de código, busque el procedimiento **ClearLog\_Click**.
2. Llame al método **Clear** en la instancia del componente **EventLog**:

```
// Visual J#
```



```
eventLog1.Clear();
```

## Comprobar registros y orígenes

En este procedimiento, creará dos procedimientos: uno que comprueba la existencia del registro personalizado y uno que comprueba la existencia de la cadena de origen. Estos procedimientos se utilizarán para comprobar los resultados de diferentes acciones que se ejecutan al ejecutar el proyecto.

### Para comprobar que el registro personalizado existe

1. En el Editor de código, busque el procedimiento **VerifyLog\_Click**.
2. Cree un cuadro de mensaje que evalúe si existe el registro de eventos especificado y muestre verdadero o falso según el resultado. Utilice este código:

```
// Visual J#  
MessageBox.Show("" + EventLog.Exists("NewLog1"));
```

### Para comprobar que el origen existe

1. En el Editor de código, busque el procedimiento **VerifySource\_Click**.
2. Cree un cuadro de mensaje que evalúe si existe el origen especificado y muestre verdadero o falso según el resultado. Utilice este código:

```
// Visual J#  
MessageBox.Show("" + EventLog.SourceExists("Source1"));
```

## Quitar orígenes

En este procedimiento, escribirá código para eliminar una cadena de origen. Para ello, comprobará en primer lugar que el origen en cuestión (Source1) existe; a continuación, llamará al método **DeleteEventSource** para quitarlo.

### Para quitar el origen de eventos creado

1. En el Editor de código, busque el procedimiento **RemoveSource\_Click**.
2. Agregue el código siguiente:

```
// Visual J#  
if (EventLog.SourceExists("Source1"))  
{  
    EventLog.DeleteEventSource("Source1");  
}
```

## Leer entradas

En este procedimiento, escribirá código para iterar en la colección de entradas del registro de eventos y mostrar los mensajes existentes en el registro.

### Para leer entradas del registro personalizado creado

1. En el Editor de código, busque el procedimiento **ReadEntry\_Click**.
2. Agregue el código siguiente:

```
// Visual J#  
int count = eventLog1.get_Entries().get_Count();  
for (int i = 0; i < count; i++)  
{  
    MessageBox.Show(eventLog1.get_Entries().get_Item(i).get_Message());  
}
```

## Probar el código

En esta sección, utilizará el Explorador de servidores para comprobar los resultados del código.

### Iniciar el Explorador de servidores

1. En el menú **Ver**, vaya al Explorador de servidores.
2. Expanda el nodo del servidor actual y, a continuación, expanda el nodo **Event Logs** situado debajo de él.

### Genere y ejecute la aplicación

1. Guarde los archivos y presione F5 para generar e iniciar el proyecto. Aparecerá el formulario con los ocho botones creados.
2. Haga clic en el botón **Crear registro personalizado**.

**Nota** Para crear registros de eventos de Windows, deberá tener los privilegios adecuados para el servidor en el que se ejecuta la aplicación. Si en este momento recibe un error de seguridad, consulte al administrador del sistema.

3. Vuelva al producto, mientras se encuentra en modo de ejecución y haga clic con el botón secundario del *mouse* (ratón) en el nodo **Event Logs** del Explorador de servidores.
4. Haga clic en **Actualizar**.
5. Compruebe que el registro **NewLog1** aparece ahora en el nodo **Registros de eventos**.

### Para probar la creación, eliminación y comprobación de registros personalizados

1. Vuelva al formulario en ejecución y haga clic en el botón **Comprobar que existe el registro**.  
Debe aparecer un mensaje indicando "verdadero".
2. Haga clic en el botón **Eliminar registro** y, a continuación, haga clic otra vez en el botón **Comprobar que existe el registro**.  
Esta vez, el mensaje debe indicar "falso".
3. Haga clic otra vez en el botón **Crear registro personalizado** para crear de nuevo el registro.

### Para probar la escritura y lectura de entradas en el registro personalizado

1. En el formulario, haga clic en el botón **Escribir entrada**.
2. Vaya al Explorador de servidores y expanda el registro **NewLog1**.
3. Expanda el nodo **Source1** que se encuentra debajo de él.  
Deberá ver que se agregaron dos entradas al registro. Una tendrá un icono que indica que se trata de una entrada informativa y la otra tendrá un icono que indica que es un error.
4. Vuelva al formulario y haga clic en el botón **Leer entrada**.  
Recibirá dos mensajes: uno que contiene la entrada informativa y otro que contiene la entrada de error.

**Nota** Si hizo clic varias veces en el botón **Escribir entrada**, habrá recibido más mensajes.

### Para probar el borrado del registro

1. En el formulario, haga clic en el botón **Borrar registro**.
2. En el Explorador de servidores, haga clic con el botón secundario del *mouse* en el nodo **NewLog1** y, a continuación, haga clic en **Actualizar**.  
Deberá ver que el registro ya no contiene entradas.

### Para probar la eliminación de la cadena de origen

1. En el formulario, haga clic en el botón **Quitar origen de eventos**.
2. Haga clic en el botón **Comprobar que existe el origen**. Debe recibir un mensaje con el texto "Falso" que indica que el origen **Source1** ya no existe.
3. Haga clic en el botón **Escribir entrada**.

**Nota** Esta acción le permitirá escribir entradas en el registro, porque el método **WriteEntry** establecerá el origen si no existe en la actualidad.

4. En el Explorador de servidores, haga clic con el botón secundario del *mouse* en el nodo **NewLog1** y, a continuación, haga clic en **Actualizar**. Debe ver dos entradas en el registro.
5. Haga clic otra vez en el botón **Comprobar que existe el origen**. Debe recibir un mensaje con el texto "verdadero" que indica que el origen **Source1** existe.
6. Si lo desea, puede hacer clic en el botón **Quitar origen de eventos** cuando termine de realizar la prueba. De esta forma quitará el origen **Source1** del Registro para que no haya cambios permanentes en la configuración del sistema.

### **Vea también**

[Tutoriales sobre la supervisión del sistema](#) | [Registrar eventos de aplicación, servidor y seguridad](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutoriales: instalar un componente de registro de eventos con Visual J#

Los procedimientos de este tema le guiarán por el proceso de configuración de un componente de instalación para una instancia del componente **EventLog**, incluida la creación de la instancia del componente, la adición de un instalador para él y la compilación y ejecución del instalador.

## Para crear el componente EventLog

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, elija **Visual J#** en el panel de la izquierda y, a continuación, elija la plantilla **Aplicación para Windows**. Asigne al proyecto el nombre **MyEventLog**.
3. En la ficha **Componentes** del Cuadro de herramientas, arrastre un componente EventLog hasta el formulario.
4. En la ventana Propiedades, defina los siguientes valores:
  - Asigne a la propiedad **Log** el valor **Application**.
  - Asigne a la propiedad **MachineName** el nombre del servidor en el que reside el registro de eventos. Use un punto (.) para el equipo local.
  - Asigne a la propiedad **Source** la cadena que desee. En este caso, puede utilizar el nombre del proyecto.
5. Guarde los archivos.

**Nota** Para obtener más información sobre cómo crear instancias del componente **EventLog** y definir sus propiedades, vea [Registrar eventos de aplicación, servidor y seguridad](#).

## Para crear un componente de instalación para el componente EventLog

1. En el diseñador, haga clic en el componente **eventLog1**.
2. En el área **Descripción**, en la parte inferior de la ventana **Propiedades**, haga clic en el vínculo **Agregar instalador**.

Aparecerá una clase **ProjectInstaller** en el proyecto, a la cual se agregará un componente de instalación. Ahora, ya puede instalar y ejecutar la aplicación.

## Para generar la instalación

- Guarde el proyecto y génerele.

**Nota** Puesto que este instalador funcionará correctamente sin cambiar los métodos existentes, no necesitará reemplazar los métodos **Install**, **Commit**, **Rollback** y **Uninstall**.

Un proyecto de instalación instalará los archivos de proyecto compilados y ejecutará los instaladores necesarios para ejecutar una aplicación para Windows.

## Para crear un proyecto de instalación para el servicio

1. En el menú **Archivo**, elija **Agregar proyecto** y, a continuación, haga clic en **Nuevo proyecto**.
2. En el panel **Tipos de proyecto**, seleccione la carpeta **Proyectos de instalación e implementación**.
3. En el panel **Plantillas**, seleccione **Proyecto de instalación**. Asigne al proyecto el nombre **MyEventLogSetup**.

Se agregará un proyecto de instalación a la solución. A continuación, deberá agregar el resultado del proyecto de aplicación para Windows a la instalación.

## Para agregar el resultado del proyecto a la instalación

1. En el Explorador de soluciones, haga clic con el botón secundario en **MyEventLogSetup**, elija **Agregar** y, a continuación, **Resultados del proyecto**.

Aparecerá el cuadro de diálogo **Agregar grupo de resultados del proyecto**.

2. Compruebe que **MyEventLog** aparece seleccionado en el cuadro **Proyecto**.
3. En el cuadro de lista, seleccione **Resultado principal** y haga clic en **Aceptar**.

Se agregará al proyecto de instalación un elemento de proyecto para el resultado principal de **MyEventLog**. Ahora,

agregue una acción personalizada para instalar el archivo MyNewService.exe.

### Para agregar una acción personalizada a la instalación

1. En el Explorador de soluciones, haga clic con el botón secundario en el proyecto de instalación, elija **Ver** y, a continuación, **Acciones personalizadas**.

Aparece el editor de **Acciones personalizadas**.

2. En el editor de **Acciones personalizadas**, haga clic con el botón secundario en el nodo **Acciones personalizadas** y elija **Agregar acción personalizada**.

Aparecerá el cuadro de diálogo **Seleccionar elemento en el proyecto**.

3. Haga doble clic en la opción **Carpeta de la aplicación**, en el cuadro de lista, para abrirla, seleccione **Resultado principal de MyEventLog (Activo)** y haga clic en Aceptar.

El resultado principal se agrega a los cuatro nodos de las acciones personalizadas: **Instalar**, **Confirmar**, **Deshacer** y **Desinstalar**.

4. Genere el proyecto de instalación.
5. Sitúese en el directorio donde se almacenó el proyecto de instalación y ejecute el archivo .msi para instalar **MyEventLog.exe**.

### Vea también

[Introducción a los componentes de instalación](#) | [Iniciar el Visor de eventos](#) | [Registrar eventos de aplicación, servidor y seguridad](#) | [Tutoriales sobre la supervisión del sistema](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutorial: crear una cola y trabajar con mensajes en Visual J#

Los procedimientos de este tema muestran el proceso de creación de una cola de mensajes mediante el componente **MessageQueue**. Con ese componente, es posible enviar mensajes a la cola y recuperarlos de ella. Los mensajes registran solicitudes de ayuda.

**Nota** Para ver información sobre colas en el Explorador de servidores, o para obtener acceso a las colas mediante programación, se debe instalar el componente de Windows **Message Queue Server** en el equipo cliente. Para agregar este servicio, utilice la herramienta **Agregar o quitar programas** del **Panel de control**.

## Para crear una cola de mensajes en el equipo

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, elija **Visual J#** en el panel de la izquierda y, a continuación, elija la plantilla **Aplicación para Windows**. Asigne el nombre **MessageQ**.
3. Abra el Explorador de servidores. Para obtener más información, vea [Acceso e inicialización del Explorador de servidores](#).
4. Expanda el nodo **Servidores**.
5. Expanda el nodo del servidor local. El nodo del servidor se identifica mediante el nombre del equipo.
6. Expanda el nodo **Colas de mensajes**.
7. Haga clic con el botón secundario del *mouse* en **Colas privadas** y, en el menú contextual, seleccione **Crear cola**.
8. Escriba **HelpRequest** como nombre para la cola. No active la casilla **Hacer la cola transaccional**.
9. Se crea una nueva cola privada, denominada **HelpRequest**, que aparece en el Explorador de servidores.

**Nota** También puede ver la cola recién creada por medio del icono **Mi PC**. Haga clic con el botón secundario del *mouse* en el icono **Mi PC** del escritorio y, en el menú contextual, elija **Administrar**. Expanda el nodo **Servicios y Aplicaciones**. Expanda el nodo **Message Queue Server** y seleccione la carpeta **Colas privadas**. La nueva cola aparece en la lista de colas.

## Para agregar un componente MessageQueue para la cola de mensajes

1. Arrastre la cola **HelpRequest** desde el Explorador de servidores hasta el formulario. Se agregará al proyecto un nuevo componente **MessageQueue** configurado para la cola **HelpRequest**.

El componente **MessageQueue** se utiliza para obtener acceso mediante programación a los mensajes contenidos en la cola **HelpRequest** creada en la sección anterior.

2. Asigne a la propiedad **Name** del componente **MessageQueue** el valor **helpRequestQueue**.
3. En la ventana Propiedades, expanda el nodo **MessageReadPropertyFilter**. Asigne a la propiedad **Priority** el valor **true**. Esto hace que se recupere la prioridad del mensaje al recuperar un mensaje de la cola.

La interfaz de usuario que se va a crear en el siguiente procedimiento permite al usuario escribir texto para una solicitud de ayuda y establecer la prioridad del mensaje. El usuario hace clic en un botón Send (Enviar) para enviar la solicitud a la cola. Una cuadrícula de datos muestra el contenido de la cola. La interfaz de usuario también incluye botones para actualizar la cuadrícula con el contenido actual de la cola y para vaciar la cola.

## Para crear la interfaz de usuario

1. Desde la ficha **Windows Forms** del Cuadro de herramientas, agregue los siguientes controles a Form1:

- Dos etiquetas
- Dos cuadros de texto
- Tres botones
- Una casilla de verificación
- Una cuadrícula de datos

2. Defina las siguientes propiedades de los controles:

Control	Propiedad	Valor nuevo
Label1	Text	Name
Label2	Text	Message
TextBox1	Name	txtName
	Text	(En blanco)

TextBox2	Name	txtMessage
	Text	(En blanco)
	Multiline	true
Button1	Name	sendMessage
	Text	Enviar mensaje
Button2	Name	refreshMessages
	Text	Actualizar lista de mensajes
Button3	Name	purgeMessages
	Text	Eliminar mensajes
CheckBox1	Name	highPriority
	Text	Prioridad alta
DataGrid1	Name	messageGrid

3. Organice los controles de forma ordenada.

### Para enviar un mensaje a la cola

1. En el diseñador, haga doble clic en el botón **sendMessage** para crear el controlador del evento **Click** en el Editor de código. El controlador tiene el siguiente aspecto:

```
// Visual J#
private void sendMessage_Click(Object sender, System.EventArgs e)
{
}
```

2. Agregue código al método para crear una nueva instancia del objeto **Message** y enviarla a la cola.

```
// Visual J#
System.Messaging.Message theMessage = new System.Messaging.Message(txtMessage.get_Text());
;
theMessage.set_Label(txtName.get_Text());
if (highPriority.get_Checked())
    theMessage.set_Priority(System.Messaging.MessagePriority.Highest);
else
    theMessage.set_Priority(System.Messaging.MessagePriority.Normal);

helpRequestQueue.Send(theMessage);
```

3. Agregue un método para mostrar el contenido de la cola en el control **DataGrid**. Este método utiliza el método **MessageQueue.GetAllMessages** para recuperar todos los mensajes de la cola. Las propiedades seleccionadas de la cola se agregan a un objeto **DataTable**, que se utiliza como origen de datos para el control **DataGrid**. Para recuperar el texto del mensaje, deberá crear un formateador para el mensaje.

```
// Visual J#
private void DisplayMessages()
{
    DataTable messageTable = new DataTable();
    messageTable.get_Columns().Add("Name");
    messageTable.get_Columns().Add("Message");
    messageTable.get_Columns().Add("Priority");
    System.Messaging.Message[] messages;
    messages = helpRequestQueue.GetAllMessages();
    System.Messaging.XmlMessageFormatter stringFormatter;
    stringFormatter = new System.Messaging.XmlMessageFormatter(
        new String[] {"System.String"});
    for (int index = 0; index < messages.length; index++)
    {
        messages[index].set_Formatter(stringFormatter);
        messageTable.get_Rows().Add(new System.String[] {
            messages[index].get_Label(),

```

```

        messages[index].get_Body().ToString(),
        messages[index].get_Priority().ToString()});
    }
    messageGrid.set_DataSource(messageTable);
}

```

4. Agregue una línea más de código al final del evento **Click** del botón **sendMessage** para actualizar la vista de mensajes.

```

// Visual J#
DisplayMessages();

```

### Para mostrar el contenido de la cola

1. En el diseñador, haga doble clic en el botón **refreshMessage** para crear el controlador del evento **Click** en el Editor de código. El controlador tiene el siguiente aspecto:

```

// Visual J#
private void refreshMessages_Click(Object sender, System.EventArgs e)
{
}

```

2. En el controlador del evento **Click**, llame al método **DisplayMessages**.

```

// Visual J#
DisplayMessages();

```

### Para vaciar el contenido de la cola

1. En el diseñador, haga doble clic en el botón **purgeMessage** para crear el controlador del evento **Click** en el Editor de código. El controlador tiene el siguiente aspecto:

```

// Visual J#
private void purgeMessages_Click(Object sender, System.EventArgs e)
{
}

```

2. En este método, llame al método **Purge** de **helpRequestQueue** y, a continuación, actualice el contenido del control **DataGrid**.

```

// Visual J#
helpRequestQueue.Purge();
DisplayMessages();

```

### Para probar la aplicación

1. Presione F5 para ejecutar la aplicación.
2. Escriba su nombre y un mensaje breve.
3. Haga clic en **Enviar mensaje** para enviar el mensaje a la cola y actualizar la presentación en pantalla.
4. Haga clic en **Purgar mensajes** para eliminar todos los mensajes de la cola. La lista de mensajes aparecerá vacía.

### Vea también

[Introducción a la mensajería](#) | [Crear colas](#) | [Crear instancias del componente MessageQueue](#) | [Enviar mensajes simples](#) | [Recuperar mensajes](#) | [Crear componentes de mensajería](#) | [Tutoriales sobre el componente de servicios Framework](#)



# Tutorial: cambiar y recuperar valores de los contadores de rendimiento en Visual J#

Los procedimientos de este tutorial muestran cómo trabajar con los valores de un contador de rendimiento mediante los métodos de la clase **PerformanceCounter**. Un contador de rendimiento es el medio que utiliza Windows para recopilar datos de rendimiento de los diversos recursos del sistema. Windows contiene un conjunto de contadores predefinidos, organizados en categorías, con los que se puede interactuar. Los valores que se recuperan de un contador pueden ser valores sin formato o valores calculados que cambian con el tiempo. Existen varias formas de aumentar y disminuir el valor actual de un contador.

En este tutorial, podrá:

- Crear y configurar un componente **PerformanceCounter** para interactuar con un contador suministrado por el sistema.
- Crear una aplicación de Windows que contiene elementos de interfaz de usuario que permiten recuperar y mostrar valores de un contador.
- Escribir código con el método **RawValue** para definir y recuperar el valor sin formato del contador.
- Escribir código con los métodos **Increment**, **IncrementBy** y **Decrement** para modificar el valor del contador y recuperar su nuevo valor.

## Para crear la aplicación para Windows

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, elija **Visual J#** en el panel de la izquierda y, a continuación, elija la plantilla **Aplicación para Windows**. Asigne al proyecto el nombre **PerformanceCounterExample**.
3. En la ficha **Windows Forms** del Cuadro de herramientas, agregue los siguientes controles a la aplicación:
  - Dos etiquetas
  - Un cuadro de texto
  - Cinco botones

4. Defina las siguientes propiedades en los controles:

Control	Propiedad	Valor
Label1	Name	<b>lblCounterValue</b>
	Text	<b>(En blanco)</b>
Label2	Name	<b>lblSystemCounterValue</b>
	Text	<b>(En blanco)</b>
Textbox1	Name	<b>txtValue</b>
	Text	<b>(En blanco)</b>
Button1	Name	<b>btnSetRawValue</b>
	Text	<b>Definir el valor sin formato del contador personalizado</b>
Button2	Name	<b>btnGetNextValue</b>
	Text	<b>Obtener el siguiente valor del contador del sistema</b>
Button3	Name	<b>btnIncrement</b>
	Text	<b>Incrementar el contador personalizado en 1</b>
Button4	Name	<b>btnDecrement</b>
	Text	<b>Disminuir el contador personalizado en 1</b>
Button5	Name	<b>btnIncrementBy</b>
	Text	<b>Incrementar el contador personalizado en un valor</b>

5. Organice los controles como desee.
6. Guarde su trabajo.

## Para crear y configurar el componente PerformanceCounter

1. Vaya al Explorador de servidores y busque el nodo **Servidores**. Para obtener más información, vea [Acceso e inicialización del Explorador de servidores](#).
2. Localice la lista correspondiente a su equipo en el nodo **Servidores** y expándalo. Podrá ver entradas para contadores de rendimiento, colas de mensajes, registros de eventos y servicios.

3. Expanda el nodo **Contadores de rendimiento** y busque el nodo **Procesador**.
4. Busque el nodo **% de tiempo de procesador** y expándalo.
5. Arrastre el contador **\_Total** hasta el formulario. Se agregará al proyecto un componente **PerformanceCounter** configurado para el contador **\_Total**. El componente se denomina **performanceCounter1**.

#### Para recuperar el valor sin formato del contador del sistema

1. En el menú **Vista**, haga clic en **Código** para abrir el Editor de código para Form1.
2. Compruebe que hay una instrucción **import** que especifique el espacio de nombres **System.Diagnostics**. Si no existe, agréguela al principio del código del formulario como se muestra a continuación.

```
// Visual J#
import System.Diagnostics.*;
```

3. En el diseñador, haga doble clic en el botón **Obtener siguiente valor** para crear el controlador del evento **Click**. Este método devuelve el valor calculado del contador, no el valor sin formato. Agregue el siguiente código para recuperar y mostrar el siguiente valor del contador que la instancia de **PerformanceCounter** está supervisando:

```
// Visual J#
private void btnGetNextValue_Click (Object sender, System.EventArgs e)
{
    lblSystemCounterValue.setText("The current value of the system counter is: " + performanceCounter1.NextValue());
}
```

Durante el resto de los procedimientos, trabajará con una categoría y un contador personalizados.

#### Para crear una categoría y un contador personalizados

1. Vaya al Explorador de servidores y busque el nodo **Servidores**. Para obtener más información, vea [Acceso e inicialización del Explorador de servidores](#).
2. Localice la lista correspondiente a su equipo en el nodo **Servidores** y expándalo.
3. Haga clic con el botón secundario del *mouse* en el nodo **Contadores de rendimiento** y seleccione **Crear categoría nueva**, en el menú contextual.

Aparecerá el cuadro de diálogo **Generador de contadores de rendimiento**.

4. Especifique **MyNewCategory** como **Nombre de categoría**.
5. Haga clic en **Nuevo** para agregar un nuevo contador, y asígnele el nombre **MyNewCounter**. Haga clic en **Aceptar** para crear la nueva categoría y el contador.

**Nota** En este paso, necesitará permiso para escribir en el Registro. Si no puede crear el contador, pida ayuda al administrador del sistema.

6. En el Explorador de servidores, seleccione el nuevo contador y arrástrelo hasta el formulario. Se agrega un nuevo componente **PerformanceCounter** al proyecto, configurado para el nuevo contador, **MyNewCounter**.

**Nota** Estos pasos crean el contador en el equipo. Si desea implementar la aplicación, deberá crear el contador en el equipo de destino. Para ello, debe agregar un instalador al proyecto. Seleccione el objeto **PerformanceCounter** y, en la ventana Propiedades, haga clic en **Agregar instalador**. Para obtener información detallada, vea [Introducción a los componentes de instalación](#).

#### Para definir el valor sin formato del contador personalizado

1. En el diseñador, seleccione **performanceCounter2** en Visual J#.
2. Asigne el valor **false** a la propiedad **ReadOnly**.
3. Haga doble clic en el botón **btnSetRawValue** para crear el controlador del evento **Click** en el Editor de código.
4. Agregue el siguiente código para definir el valor sin formato del contador creado:

```
// Visual J#
private void btnSetRawValue_Click (Object sender, System.EventArgs e)
{
```

```
performanceCounter2.set_RawValue(Long.parseLong(txtValue.getText()));  
}
```

5. Agregue el siguiente código para mostrar el valor del contador en el primero de los controles de etiqueta (label):

```
// Visual J#  
lblCounterValue.setText("" + performanceCounter2.NextValue());
```

El procedimiento debe tener el siguiente aspecto:

```
// Visual J#  
private void btnSetRawValue_Click (Object sender, System.EventArgs e)  
{  
    performanceCounter2.set_RawValue(Long.parseLong(txtValue.getText()));  
    lblCounterValue.setText("" + performanceCounter2.NextValue());  
}
```

### Para incrementar en uno el valor del contador personalizado y mostrarlo

1. En el diseñador, haga doble clic en el botón **Incrementar el contador personalizado en 1** para crear el controlador del evento **Click** en el Editor de código.
2. Agregue el siguiente código para sumar uno al contador personalizado:

```
// Visual J#  
private void btnIncrement_Click (Object sender, System.EventArgs e)  
{  
    performanceCounter2.Increment();  
}
```

3. Agregue el siguiente código para mostrar el valor del contador en el primero de los controles de etiqueta (label):

```
// Visual J#  
lblCounterValue.setText("" + performanceCounter2.NextValue());
```

El procedimiento debe tener el siguiente aspecto:

```
// Visual J#  
private void btnIncrement_Click (Object sender, System.EventArgs e)  
{  
    performanceCounter2.Increment();  
    lblCounterValue.setText("" + performanceCounter2.NextValue());  
}
```

### Para disminuir en uno el valor del contador personalizado y mostrarlo

1. En el diseñador, haga doble clic en el botón **Disminuir el contador personalizado en 1** para crear el controlador del evento **Click** en el Editor de código.
2. Agregue el siguiente código para restar uno al contador personalizado:

```
// Visual J#  
private void btnDecrement_Click (Object sender, System.EventArgs e)  
{  
    performanceCounter2.Decrement();  
}
```

3. Agregue el siguiente código para mostrar el valor del contador en el primero de los controles de etiqueta (label):

```
// Visual J#
```

```
lblCounterValue.setText("" + performanceCounter2.NextValue());
```

El procedimiento debe tener el siguiente aspecto:

```
// Visual J#
private void btnDecrement_Click (Object sender, System.EventArgs e)
{
    performanceCounter2.Decrement();
    lblCounterValue.setText("" + performanceCounter2.NextValue());
}
```

### Para incrementar el contador personalizado en un valor definido por el usuario

1. En el diseñador, haga doble clic en el botón **Incrementar el contador personalizado en un valor** para crear el controlador del evento **Click** en el Editor de código.
2. Agregue el siguiente código para incrementar el valor del contador con el valor especificado en el cuadro de texto:

```
// Visual J#
private void btnIncrementBy_Click(Object sender, System.EventArgs e)
{
    performanceCounter2.IncrementBy(Long.parseLong(txtValue.getText()));
}
```

3. Agregue el siguiente código para mostrar el valor del contador en el primero de los controles de etiqueta (label):

```
// Visual J#
lblCounterValue.setText("" + performanceCounter2.NextValue());
```

El procedimiento debe tener el siguiente aspecto:

```
// Visual J#
private void btnIncrementBy_Click (Object sender, System.EventArgs e)
{
    performanceCounter2.IncrementBy(Long.parseLong(txtValue.getText()));
    lblCounterValue.setText("" + performanceCounter2.NextValue());
}
```

### Para probar la aplicación

1. Guarde los archivos.
2. Presione F5 para compilar e iniciar la aplicación.
3. Haga clic en el botón **Obtener el siguiente valor del contador del sistema** para recuperar el valor actual del contador de la categoría Procesador. Puesto que va a recuperar el valor mediante el método **NextValue**, la primera llamada devolverá 0.

Podrá ver el valor actual mostrado en la etiqueta.

4. Escriba **25** en el cuadro de texto y haga clic en **Definir el valor sin formato del contador personalizado**.

El campo de etiqueta debería actualizarse para indicar que el valor sin formato es ahora de 25.

5. Haga clic en el botón **Incrementar el contador personalizado en 1**.

El valor de la etiqueta debería incrementarse en 1.

6. Haga clic en el botón **Disminuir el contador personalizado en 1**.

El valor de la etiqueta debería disminuir en 1.

7. Escriba **25** en el cuadro de texto y haga clic en **Incrementar el contador personalizado en un valor**.

El valor de la etiqueta debería incrementarse en 25.

También puede ver el contador de rendimiento en la herramienta Rendimiento de Windows.

#### **Para ver el contador de rendimiento en la herramienta Rendimiento de Windows**

1. En la barra de herramientas, haga clic en el botón **Inicio**, seleccione **Configuración** y, a continuación, haga clic en **Panel de control**.
2. En **Panel de control**, haga doble clic en **Herramientas administrativas** y, a continuación, haga doble clic en **Rendimiento**.
3. Haga clic con el botón secundario en la lista de contadores bajo el gráfico de rendimiento y, en el menú contextual, elija **Agregar contadores**.
4. Seleccione **MyNewCategory** en la lista de **Objeto de rendimiento** y **MyNewCounter** en la lista de contadores. Haga clic en **Agregar** para finalizar.
5. Haga clic en **Ver informe**, en la barra de herramientas, para mostrar el valor del contador.

#### **Vea también**

[Supervisar umbrales de rendimiento](#) | [Introducción a la supervisión de umbrales de rendimiento](#) | [Tutoriales sobre los contadores de rendimiento](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutorial: recuperar categorías y contadores con Visual J#

Los procedimientos de este tutorial le guiarán a través del proceso de crear y configurar instancias del componente **PerformanceCounterCategory** y de utilizarlos para recuperar listas de categorías de contadores de rendimiento y de contadores del sistema. Un contador de rendimiento es el medio que utiliza Windows para recopilar datos de rendimiento de los diversos recursos del sistema. Windows contiene un conjunto de contadores predefinidos, organizados en categorías, con los que se puede interactuar. Cada categoría y cada contador están relacionados con un área específica de funcionalidad del sistema.

En este tutorial:

- Creará una instancia del componente **PerformanceCounterCategory** y la configurará de modo que interactúe con una categoría específica de contadores generados por el sistema.
- Creará una aplicación para Windows que muestra información acerca de las categorías y contadores en un cuadro de lista.
- Utilizará el método **GetCategories** para devolver una lista de categorías del equipo local.
- Utilizará el método **GetCounters** para devolver una lista de contadores de la categoría especificada.

## Para crear la aplicación para Windows

1. En el cuadro de diálogo **Nuevo proyecto**, cree una **aplicación de Visual J# para Windows**. Asigne el nombre **PerformanceCounterCategoryExample** al proyecto.
2. En la ficha **Windows Forms** del Cuadro de herramientas, agregue dos botones y dos cuadros de lista al formulario. Organícelos en el orden que desee y establezca las siguientes propiedades:

Control	Propiedad	Valor
Button1	Name	<b>btnGetCounters</b>
	Text	<b>Obtener contadores</b>
Button2	Name	<b>btnGetCategories</b>
	Text	<b>Obtener categorías</b>
ListBox1	Name	<b>lstCounters</b>
	ScrollAlwaysVisible	<b>True</b>
ListBox2	Name	<b>lstCategories</b>
	ScrollAlwaysVisible	<b>True</b>

3. Guarde su trabajo.

## Para recuperar la lista de categorías

1. En la vista Diseño, haga doble clic en el botón **Obtener categorías** para tener acceso al Editor de código. El cursor se colocará en el evento **btnGetCategories\_Click**.
2. En la parte superior de la pantalla, agregue una instrucción **import** que haga referencia al espacio de nombres **System.Diagnostics**.

```
// Visual J#
import System.Diagnostics.*;
```

3. En el procedimiento **btnGetCategories\_Click**, agregue el código siguiente para recuperar la lista de categorías del equipo local.

```
// Visual J#
private void btnGetCategories_Click (Object sender, System.EventArgs e)
{
    PerformanceCounterCategory[] myCat2;

    // Remove the current contents of the list.
    lstCategories.getItems().Clear();

    // Retrieve the categories.
    myCat2 = PerformanceCounterCategory.GetCategories();

    // Add the retrieved categories to the list.
    for (int i = 0; i < myCat2.length; i++)
```

```

    {
        lstCategories.getItems().Add(myCat2[i].get_CategoryName());
    }
}

```

### Para recuperar la lista de contadores

1. En la vista Diseño, haga doble clic en el botón **Obtener contadores** para tener acceso al Editor de código. El punto de inserción se colocará en el evento **btnGetCounters\_Click**.
2. Agregue el código siguiente para recuperar la lista de contadores de la categoría seleccionada.

```

// Visual J#
private void btnGetCounters_Click (Object sender, System.EventArgs e)
{
    String[] instanceNames;
    System.Collections.ArrayList counters = new System.Collections.ArrayList();
    if (this.lstCategories.get_SelectedIndex() != -1)
    {
        PerformanceCounterCategory mycat = new PerformanceCounterCategory(lstCategories.get
_SelectedItem().ToString());

        // Remove the current contents of the list.
        lstCounters.getItems().Clear();

        // Retrieve the counters.
        try
        {
            instanceNames = mycat.GetInstanceNames();
            if (instanceNames.length == 0)
            {
                counters.AddRange(mycat.GetCounters());
            }
            else
            {
                for (int i = 0; i < instanceNames.length; i++)
                {
                    counters.AddRange(mycat.GetCounters(instanceNames[i]));
                }
            }

            // Add the retrieved counters to the list.
            int cnt = counters.get_Count();
            for (int i = 0; i < cnt; i++)
            {
                lstCounters.getItems().Add(((PerformanceCounter) counters.get_Item(i)).get_Counter
Name());
            }
        }
        catch (System.Exception ex)
        {
            MessageBox.Show("Unable to list the counters for this category:\n" + ex.get_Mess
age());
        }
    }
}

```

### Para probar la aplicación

1. Guarde y compile la aplicación.

2. Haga clic en el botón **Obtener categorías**. Verá una lista de categorías en el cuadro de lista.
3. Seleccione una de las categorías y haga clic en el botón **Obtener contadores**. Verá una lista de contadores para la categoría seleccionada.

### **Vea también**

[Supervisar umbrales de rendimiento](#) | [Introducción a la supervisión de umbrales de rendimiento](#) | [Tutoriales sobre los contadores de rendimiento](#) | [Tutoriales sobre el componente de servicios Framework](#)



# Tutorial: administrar un proceso de Windows con Visual J#

Los procedimientos de este tema le guiarán por los pasos necesarios para crear un proceso, responder cuando se detiene un proceso y detener procesos. En la primera sección, creará una aplicación de Windows con controles **Button** para iniciar y detener un proceso **Notepad**. Iniciará varias instancias de **Notepad** por separado y, a continuación, las detendrá como grupo. En la segunda sección, creará una aplicación de consola que enumera los procesos que se ejecutan en el equipo.

Para obtener más información sobre cómo usar el componente **Process** para interactuar con procesos del sistema, vea [Introducción a la supervisión y administración de procesos de Windows](#).

## Para crear la aplicación

1. En el cuadro de diálogo **Nuevo proyecto**, cree una **aplicación de Visual J# para Windows**. Asigne el nombre **ProcessExample** al proyecto.
2. Con el diseñador de Form1 abierto, haga clic en la ficha **Windows Forms** del Cuadro de herramientas y, a continuación, agregue dos botones al formulario.
3. En la ventana Propiedades, cambie las propiedades siguientes:

Control	Propiedad	Valor
Button1	Name	<b>ButtonStart</b>
	Text	<b>Iniciar proceso</b>
Button2	Name	<b>ButtonStop</b>
	Text	<b>Detener proceso</b>

4. Haga clic en la ficha **Componente** del Cuadro de herramientas y, a continuación, arrastre una instancia del componente **Process** a la superficie del diseñador.
5. Asigne al componente el nombre **MyProcess**.

## Para iniciar el proceso Notepad.exe

1. En la ventana Propiedades de la instancia del componente **myProcess**, expanda la propiedad **StartInfo** y establezca la propiedad **FileName** en **notepad.exe**.
2. Haga doble clic en el botón **Inicio** para tener acceso al Editor de código y, a continuación, agregue el código siguiente al evento **ButtonStart\_Click**:

```
// Visual J#
myProcess.Start();
```

3. Guarde todos los archivos y, a continuación, genere y ejecute la aplicación.
4. Haga clic en el botón **Iniciar proceso** varias veces. Verá instancias separadas del Bloc de notas para cada clic.
5. Cierre cada una de las aplicaciones **Bloc de notas**.
6. Cierre la aplicación **Form1**.

## Para cerrar todas las instancias actuales del proceso Notepad.exe

1. Vaya al Editor de código para Form1.
2. Agregue las siguientes líneas de código al principio del archivo justo después de la instrucción **package**:

```
// Visual J#
import System.Threading.*;
import System.Diagnostics.*;
```

3. En la vista Diseño, haga doble clic en el botón **Detener** para tener acceso al procedimiento **ButtonStop\_Click**.
4. Agregue el código siguiente para cerrar las instancias actuales del Bloc de notas.

```
// Visual J#
private void ButtonStop_Click (Object sender, System.EventArgs e)
{
    System.Diagnostics.Process[] myProcesses;
    myProcesses = myProcess.GetProcessesByName("Notepad");
    int count = myProcesses.length;
```

```

        for (int i = 0; i < count; i++)
        {
            System.Diagnostics.Process instance = myProcesses[i];
            instance.CloseMainWindow();
        }
    }
}

```

5. Guarde todos los archivos y, a continuación, genere y ejecute la aplicación.
6. Haga clic en el botón **Iniciar proceso** para iniciar varias instancias del Bloc de notas.
7. Haga clic en el botón **Detener proceso** para cerrar inmediatamente todas las instancias del Bloc de notas en ejecución.
8. Cierre la aplicación **Form1**.

### Para configurar el componente para que espere a que termine de ejecutarse el proceso del Bloc de notas

En este procedimiento, detendrá el código hasta que el proceso termine.

1. Vaya al Editor de código para Form1.
2. Agregue la siguiente línea de código al procedimiento **ButtonStop\_Click** antes de la llamada al método **Process.CloseMainWindow**:

```

// Visual J#
instance.WaitForExit(3000);

```

El procedimiento **ButtonStop\_Click()** deberá tener el aspecto siguiente:

```

// Visual J#
private void ButtonStop_Click (Object sender, System.EventArgs e)
{
    System.Diagnostics.Process[] myProcesses;
    myProcesses = myProcess.GetProcessesByName("Notepad");
    int count = myProcesses.length;
    for (int i = 0; i < count; i++)
    {
        System.Diagnostics.Process instance = myProcesses[i];
        instance.WaitForExit(3000);
        instance.CloseMainWindow();
    }
}

```

3. Guarde todos los archivos y, a continuación, genere y ejecute la aplicación.
4. Haga clic en el botón **Iniciar procesos** para iniciar varias instancias del Bloc de notas.
5. Haga clic en el botón **Detener procesos** para cerrar los procesos, como hizo anteriormente.

Observará que la aplicación espera ahora 3 segundos a que se detenga cada proceso antes de cerrar la siguiente instancia del proceso.

6. Cierre la aplicación **Form1**.

En esta parte del tutorial, creará una nueva aplicación de controla que recuperará y mostrará una lista de los procesos del equipo local.

### Para enumerar los procesos del equipo

1. Cree un proyecto nuevo de **Aplicación de consola**.
2. Abra el Editor de código para el módulo y, a continuación, agregue las líneas de código siguientes al principio del archivo, justo después de la instrucción **package**:

```

// Visual J#
import System.Threading.*;
import System.Diagnostics.*;

```

3. Agregue el siguiente código al módulo para enumerar los procesos del equipo:

```
// Visual J#
public static void main(String[] args)
{
    System.Diagnostics.Process[] processes;
    System.Diagnostics.Process myProcess = null;
    processes = ((System.Diagnostics.Process)myProcess).GetProcesses();
    int count = processes.length;
    for (int n = 0; n < count; n++)
    {
        System.Diagnostics.Process instance = processes[n];
        System.Console.WriteLine(instance.get_ProcessName());
    }
    System.Threading.Thread.Sleep(5000);
}
```

4. Guarde todos los archivos y, a continuación, genere y ejecute la aplicación.

La aplicación abrirá una ventana de consola con una lista de todos los procesos actualmente en ejecución en el equipo. La llamada a **Thread.Sleep** hace una pausa en la consola de cinco segundos antes de cerrarla.

**Nota** Puede que reciba excepciones en estos dos procedimientos si llama a **GetProcesses** o **GetProcessesByName** y uno de los procesos devueltos termina antes de hacer la siguiente llamada. En este caso, recibirá una excepción que indica que el proceso no existe.

#### **Vea también**

[Administrar procesos](#) | [Iniciar procesos](#) | [Detener procesos](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutorial: reaccionar a eventos del sistema de archivos con Visual J#

Los procedimientos de este tema le guiarán a través del proceso de crear un componente **FileSystemWatcher**, apuntarlo a un directorio del equipo local y, a continuación, utilizar la propiedad **Filter** para inspeccionar sólo los cambios que se produzcan en los archivos de texto. Creará un controlador de eventos que responde cuando se producen los eventos **Created** y **Changed** y utilizará un formulario para mostrar las notificaciones que resultan de estos eventos.

## Cree el formulario y los componentes necesarios para la aplicación

En este procedimiento, creará un formulario Windows Forms y una instancia del componente **FileSystemWatcher**, que colaborarán para reaccionar a eventos del nivel de directorio.

1. En el cuadro de diálogo **Nuevo proyecto**, cree una **aplicación para Windows de Visual J#** y asígnele el nombre **MyWatcher**.
2. En la ficha **Windows Forms** del Cuadro de herramientas, arrastre dos controles **Label** a la superficie del diseñador.
3. Haga clic en la ficha **Componentes** del Cuadro de herramientas y, a continuación, arrastre el icono **FileSystemWatcher** a la superficie del diseñador para el componente. De forma predeterminada, este componente se denomina **fileSystemWatcher1**.

## Para establecer propiedades para el componente FileSystemWatcher

En este procedimiento, establecerá varias propiedades para el componente, para determinar qué inspeccionará. Estas configuraciones hacen que el componente inspeccione el directorio especificado en el equipo local y que inspeccionen la creación de archivos que terminen con la extensión de nombre de archivo .txt.

1. Haga clic en el componente **FileSystemWatcher** creado en el procedimiento anterior y vea sus propiedades en la ventana Propiedades.
2. Establezca el nombre del componente en **myWatcher**.
3. Utilice la propiedad **Path** para establecer el componente **FileSystemWatcher** para que inspeccione un directorio del equipo local. Por ejemplo, en un equipo en el que se ejecute Microsoft® Windows® 2000, podría escribir lo siguiente en la propiedad **Path** para establecer el componente para que inspeccione el directorio My Documents:

```
C:\Documents and Settings\yourusername\My Documents\
```

**Sugerencia** Para este ejemplo puede utilizar cualquier directorio que desee en el equipo local.

4. Establezca la propiedad **Filter** en **\*.txt** para hacer que el componente inspeccione sólo los archivos que acaben en la extensión de nombre de archivo .txt.

## Escriba el código necesario para el componente

En este procedimiento, definirá dos controladores de eventos para el componente, que definen el procesamiento que debe tener lugar siempre que se produzcan los eventos **Changed** y **Created**.

1. Abra el Editor de código para el formulario.
2. Agregue la línea siguiente en la parte superior de la ventana, para obtener acceso a las clases **FileSystemWatcher**:

```
// Visual J#
import System.IO.*;
```

3. Vuelva a la vista Diseño y haga doble clic en el componente **FileSystemWatcher**. Aparecerá el Editor de código y se abrirá un controlador de eventos predeterminado para el evento **Changed**.

**Nota** Para obtener más información, vea [Control de eventos en formularios Windows Forms](#).

4. Utilice el código siguiente para mostrar una cadena de texto simple que comprobará que se produjo el evento:

```
// Visual J#
private void myWatcher_Changed (Object sender, System.IO.FileSystemEventArgs e)
{
```

```
        label1.setText("Changes made to: " + e.get_FullPath());  
    }
```

5. Cree el controlador de eventos para el evento **Created** que especifique lo que debe hacer la aplicación cuando el componente produzca este evento. Para obtener información acerca de la creación de controladores de eventos, vea [Crear controladores de eventos en el Diseñador de Windows Forms](#). Agregue el código que aparece a continuación para mostrar la ruta de acceso completa del archivo recién creado.

```
// Visual J#  
private void myWatcher_Created(Object sender, System.IO.FileSystemEventArgs e)  
{  
    label2.setText("The file: " + e.get_FullPath() + " has been added to your directory."  
);  
}
```

6. Guarde todos los archivos y, a continuación, genere y ejecute la aplicación.

### Para probar el componente **FileSystemWatcher**

En este procedimiento, hará modificaciones de forma manual en el directorio que está inspeccionando el componente, para forzar al controlador de eventos a que produzca los eventos **Changed** y **Created**.

1. Ejecute la aplicación que creó en el procedimiento anterior.
2. Utilice el Explorador de Windows para buscar el directorio que estableció que inspeccionara el componente **FileSystemWatcher**.
3. Abra el Bloc de notas y cree un nuevo archivo de texto. Guarde este archivo en el directorio que encontró en el paso 2 y cierre el archivo.

**Nota** Esto debe producir los eventos **Created** y **Changed**, y ejecutar el controlador definido.

4. Vuelva al formulario. Deberá ver el mensaje de creación en la etiqueta.
5. Abra el archivo que creó, escriba algunas líneas de texto y guárdelo otra vez.

**Nota** Esto debe producir el evento **Changed** y ejecutar el controlador definido.

6. Vuelva al formulario. Deberá ver el mensaje de cambio en la etiqueta.

### Vea también

[Crear instancias de componentes FileSystemWatcher](#) | [Introducción a la supervisión de eventos del sistema de archivos](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutorial: agregar objetos Active Directory con Visual J#

Los procedimientos incluidos en este tema muestran el proceso de uso del componente **DirectoryEntry** para enumerar los usuarios, grupos y servicios del equipo local. El componente **DirectoryEntry** utiliza la tecnología Active Directory para hacer este trabajo. Cada entrada creada por el componente contiene una lista de sus propiedades.

## Para crear la interfaz de usuario

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, elija **Visual J#** en el panel de la izquierda y, a continuación, elija la plantilla **Aplicación para Windows**. Asigne al proyecto el nombre **ActiveDirectory**.
3. En la ficha **Windows Forms** del Cuadro de herramientas, arrastre un control **TreeView** a Form1:
4. Establezca la propiedad **Name** del control **TreeView** en **viewPC**.

El control **TreeView** se modificará más adelante en el tutorial, de forma que contenga tres nodos de nivel superior para los usuarios, grupos y servicios respectivamente. Cada nodo de segundo nivel representa a un usuario, grupo o servicio registrado en el equipo. Cada usuario, grupo y servicio tendrá dos nodos secundarios, uno para su ruta de acceso de Active Directory y otro para sus propiedades.

## Para configurar un componente DirectoryEntry

1. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en el proyecto y elija **Agregar referencia** en el menú contextual.
2. Agregue una referencia a **System.DirectoryServices.dll**.
3. En la ficha **Componentes** del Cuadro de herramientas, arrastre un componente **DirectoryEntry** hasta el diseñador.
4. Establezca la propiedad **Name** del componente **DirectoryEntry** en **entryPC**.
5. Establezca la propiedad **Path** del componente **DirectoryEntry** en **WinNT://Dominio/NombreDeSuEquipo**. Utilice su dominio y el nombre de su equipo para indicar al componente **DirectoryEntry** que examine el equipo local mediante el proveedor de servicios de WinNT correspondiente a Active Directory.

**Nota** Si no conoce el nombre del dominio o del equipo, consulte al administrador del sistema.

## Para agregar los nodos de nivel superior al control TreeView

1. En el diseñador, haga doble clic en **Form1** para crear el controlador de evento **Form\_Load** en el Editor de código.
2. Agregue esta instrucción al principio del archivo para incluir el espacio de nombres **System.DirectoryServices** en el código. Debería ir después de la instrucción **package**.

```
// Visual J#
import System.DirectoryServices.*;
```

3. Agregue el controlador del evento **Load** para crear los tres nodos de nivel superior para usuarios, grupos y servicios. El código presenta el siguiente aspecto:

```
// Visual J#
private void Form1_Load (Object sender, System.EventArgs e)
{
    TreeNode users = new TreeNode("Users");
    TreeNode groups = new TreeNode("Groups");
    TreeNode services = new TreeNode("Services");
    viewPC.get_Nodes().AddRange(new TreeNode[] { users, groups, services });
}
```

Cuando la aplicación se ejecuta, la propiedad **Children** del componente denominado **entryPC** contiene todas las entradas encontradas en **Path** (su equipo). Cada objeto secundario es también un objeto **DirectoryEntry**. Puesto que está utilizando el proveedor de WinNT en una ruta de acceso de equipo, la propiedad **SchemaClassName** del objeto secundario **DirectoryEntry** tendrá uno de entre los valores "User", "Group" o "Service". Se puede utilizar una instrucción if para determinar el tipo de entrada y agregar un nodo al grupo correcto.

## Para agregar los nodos de entrada al control TreeView

1. Agregue este código a **Form1\_Load**, al final del código agregado anteriormente, para enumerar los objetos **DirectoryEntry** contenidos en la instancia de **entryPC**.

```
// Visual J#
System.Collections.IEnumerator enum1 = ((System.DirectoryServices.DirectoryEntries)entryPC.get_Children()).GetEnumerator();
while(enum1.MoveNext())
{
    DirectoryEntry child =(System.DirectoryServices.DirectoryEntry) enum1.get_Current();
    // Code added here in next steps.
}
```

2. Dentro del código del paso 1, agregue el siguiente código para crear un nuevo nodo para la entrada del directorio secundario y agregarlo al nodo de nivel superior apropiado.

```
// Visual J#
TreeNode newNode = new TreeNode(child.get_Name());
String name = child.get_SchemaClassName();
if (name.equals("User"))
{
    users.get_Nodes().Add(newNode);
}
else if(name.equals("Group"))
{
    groups.get_Nodes().Add(newNode);
}
else if(name.equals("Service"))
{
    services.get_Nodes().Add(newNode);
}
```

3. Agregue la siguiente instrucción después de la instrucción else if para agregar nodos para la ruta de acceso y las propiedades de la entrada. El método `AddPathAndProperties`, que deberá escribir, agrega nodos para la ruta de acceso y las propiedades de la entrada.

```
// Visual J#
AddPathAndProperties(newNode, child);
```

**Nota** En la siguiente sección se agregará el método `AddPathAndProperties`.

El método **Form\_Load** completo presenta el siguiente aspecto:

```
// Visual J#
private void Form1_Load (Object sender, System.EventArgs e)
{
    TreeNode users = new TreeNode("Users");
    TreeNode groups = new TreeNode("Groups");
    TreeNode services = new TreeNode("Services");
    viewPC.get_Nodes().AddRange(new TreeNode[] { users, groups, services });

    System.Collections.IEnumerator enum1 = ((System.DirectoryServices.DirectoryEntries)entryPC.get_Children()).GetEnumerator();
    while(enum1.MoveNext())
    {
        DirectoryEntry child =(System.DirectoryServices.DirectoryEntry) enum1.get_Current();

        ;

        TreeNode newNode = new TreeNode(child.get_Name());
        String name = child.get_SchemaClassName();
        if (name.equals("User"))
```

```

        {
            users.get_Nodes().Add(newNode);
        }
        else if(name.equals("Group"))
        {
            groups.get_Nodes().Add(newNode);
        }
        else if(name.equals("Service"))
        {
            services.get_Nodes().Add(newNode);
        }
        AddPathAndProperties(newNode, child);
    }
}

```

El componente **DirectoryEntry** tiene una propiedad **Propiedades** que contiene una colección de propiedades indizada por cadenas. El contenido de esta colección depende del esquema del componente **DirectoryEntry**. La colección **Properties** tiene una propiedad **PropertyNames**, que es una colección de todos los nombres de las propiedades. Puede recuperar cada propiedad de la colección si la recorre paso a paso. Cada propiedad dispone también de una colección de valores asociados. En este tutorial, sólo se recupera el primer valor de cada propiedad.

### Para mostrar la ruta de acceso y las propiedades de la entrada

1. Agregue el código siguiente a la clase Form1 para crear el método `AddPathAndProperties`. El parámetro `entry` representa una entrada del directorio, ya sea un usuario, un grupo o un servicio. El parámetro `node` representa el objeto **TreeNode** asociado a dicha entrada.

```

// Visual J#
private void AddPathAndProperties(TreeNode node, DirectoryEntry entry)
{
}

```

2. Agregue este código para crear un nuevo nodo que muestre la ruta de acceso de la entrada:

```

// Visual J#
node.get_Nodes().Add(new TreeNode("Path: " + entry.get_Path()));

```

3. Agregue este código para crear un nodo principal para las propiedades:

```

// Visual J#
TreeNode propertyNode = new TreeNode("Properties");
node.get_Nodes().Add(propertyNode);

```

4. Agregue este código para crear un nuevo nodo para cada propiedad de la entrada. Con cada propiedad, se muestran su nombre y valor:

```

// Visual J#
System.Collections.ICollection params = entry.get_Properties().get_PropertyNames();
System.Collections.IEnumerator enum2 = params.GetEnumerator();
while (enum2.MoveNext())
{
    String oneNode =(String) enum2.get_Current();
    ((System.DirectoryServices.PropertyValueCollection)entry.get_Properties().get_Item(oneNode)).get_Item(0).ToString();
    propertyNode.get_Nodes().Add(new TreeNode(oneNode));
}

```

El método completo presenta el siguiente aspecto:



```
// Visual J#
private void AddPathAndProperties(TreeNode node, DirectoryEntry entry)
{
    node.get_Nodes().Add(new TreeNode("Path: " + entry.get_Path()));
    TreeNode propertyNode = new TreeNode("Properties");
    node.get_Nodes().Add(propertyNode);
    System.Collections.ICollection params = entry.get_Properties().get_PropertyNames();
    System.Collections.IEnumerator enum2 = params.GetEnumerator();
    while (enum2.MoveNext())
    {
        String oneNode =(String) enum2.get_Current();
        ((System.DirectoryServices.PropertyValueCollection)entry.get_Properties().get_Item(
oneNode)).get_Item(0).ToString();
        propertyNode.get_Nodes().Add(new TreeNode(oneNode));
    }
}
```

### Para ejecutar el programa

1. Presione F5 para ejecutar el programa.
2. Abra los nodos para examinar las rutas de acceso y las propiedades. Los usuarios comparten todos el mismo conjunto de propiedades, ya que todos ellos comparten el mismo esquema. Lo anterior también es cierto para los nodos de grupos y servicios.

### Pasos siguientes

Este tutorial ha utilizado el proveedor de servicios de WinNT para Active Directory. Existen otros servicios disponibles, como LDAP (*Lightweight Directory Access Protocol*), NDS (*Novell NetWare Directory Service*) y NWCOMPAT (*Novell Netware 3.x Service*). Cada proveedor proporciona un conjunto diferente de objetos que permiten examinar y manipular directorios.

El componente **DirectorySearcher** se puede utilizar para buscar entradas desde una ruta de acceso raíz. El componente **DirectorySearcher** trabaja con el proveedor LDAP.

### Vea también

[Crear componentes de Active Directory](#) | [Información básica sobre tecnología Active Directory](#) | [DirectoryEntry \(Clase\)](#) | [DirectorySearcher \(Clase\)](#) | [Tutoriales sobre el componente de servicios Framework](#)

# Tutorial sobre propiedades dinámicas

El tutorial de esta sección muestra las propiedades dinámicas, una característica que permite almacenar algunos de los valores de propiedad de una aplicación en un archivo externo y cargarlos cuando se ejecute la aplicación.

## En esta sección

### [Almacenar y recuperar propiedades dinámicas con Visual J#](#)

Describe los principales pasos que hay que seguir para almacenar y recuperar valores de propiedades.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, componentes y controles, trabajar con Windows Forms y Web Forms, y desarrollar tareas relacionadas con datos.

### [Configurar aplicaciones utilizando propiedades dinámicas](#)

Sección de Visual Basic/Visual C# que explica cómo configurar los proyectos para que algunas de sus propiedades se almacenen en un archivo de configuración en lugar de en el código compilado.

# Tutorial: almacenar y recuperar propiedades dinámicas con Visual J#

Los procedimientos de este tema muestran el proceso de creación de un proyecto: se almacenarán algunas de sus propiedades en un archivo externo y, a continuación, se modificarán los valores almacenados y se actualizarán dinámicamente en la aplicación implantada.

En esta aplicación, se creará una instancia del componente EventLog (registro de eventos) y se almacenarán los valores de sus propiedades en un archivo.

## Para crear una instancia del componente

1. En el menú **Archivo**, elija **Nuevo** y, a continuación, haga clic en **Proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, elija **Proyectos de Visual J#** en el panel de la izquierda y, a continuación, elija la plantilla **Aplicación para Windows**. Asigne el nombre **MyLog**.
3. En la ficha **Componentes** del Cuadro de herramientas, arrastre un componente **EventLog** hasta el formulario.

**Nota** Las propiedades dinámicas no están asociadas de ningún modo con el componente **EventLog**; simplemente, utilizamos este componente en el tutorial como ejemplo del tipo de componente cuyas propiedades se pueden almacenar externamente.

4. En la ventana Propiedades del componente **EventLog**, defina los siguientes valores:
  - Asigne a la propiedad **Log** el valor **Application**, de modo que el componente pueda conectarse con un registro de eventos de Windows válido.
  - Asigne a la propiedad **MachineName** el nombre del servidor en el que reside el registro de eventos. Use un punto (.) para el equipo local.
  - Asigne a la propiedad **Source** la cadena que desee. En este caso, puede utilizar el nombre del proyecto.
5. En el menú **Ver**, haga clic en **Código** para abrir el Editor de código de Form1.
6. Expanda el nodo denominado **Código generado por el diseñador de Windows Forms**.

Aparecerá una línea con el valor de la propiedad **Log**, como la siguiente:

```
// Visual J#
this.eventLog1.set_Log("Application");
```

7. Guarde los archivos.

**Nota** Para obtener más información sobre cómo crear instancias del componente **EventLog** y definir sus propiedades, vea [Registrar eventos de aplicación, servidor y seguridad](#).

## Para mostrar el valor de la propiedad log

1. En el diseñador, arrastre una etiqueta y un botón desde la ficha **Windows Forms** del Cuadro de herramientas hasta Form1.
2. Haga doble clic en el botón para abrir el Editor de código y obtener acceso al controlador del evento **Click** del control **Button**.
3. Agregue el código siguiente al método:

```
// Visual J#
label1.set_Text("You are connected to the " + eventLog1.get_Log() + " Log.");
```

## Para configurar la propiedad Log para el componente

1. En la ventana Propiedades del componente **EventLog**, haga clic en el signo más (+) situado al lado de **Dynamic Properties**.
2. Seleccione el botón **Elipsis** que aparece junto a la propiedad **Log**.
3. En el cuadro de diálogo **Propiedades dinámicas** que aparece, seleccione la casilla de verificación **Asignar la propiedad a una clave en el archivo de configuración** y haga clic en **Aceptar**.

Esto indica que el valor de la propiedad **Log** se debería almacenar mediante la clave predeterminada: el nombre de la clase y el nombre de la propiedad. En este caso, la clave será **eventLog1.Log**.

- Abra el Editor de código para Form1 y busque el procedimiento **InitializeComponent** para **MyLog**.

Podrá ver que el código ha cambiado automáticamente para reflejar que el valor está ahora almacenado de forma externa:

```
// Visual J#
this.eventLog1.set_Log(((String)(configurationAppSettings.GetValue(("eventLog1.Log"),
    String.class.ToType()))));
this.eventLog1.set_Source("MyLog");
```

- Guarde los archivos y genere el proyecto.
- En el Explorador de soluciones, haga doble clic en el archivo .config para ver el proyecto en el editor. Encontrará la entrada para el valor de la propiedad **EventLog1.Log**:

```
// Visual J#
<add key="eventLog1.Log" value="Application" />
```

- Ejecute la aplicación. Cuando haga clic en el botón, la etiqueta indicará que se encuentra conectado al registro de la aplicación.
- Cierre la aplicación.

Asimismo, puede modificar los valores guardados aparte de ejecutar la aplicación. Puede hacerlo para definir los valores iniciales de una aplicación.

#### Para modificar el valor almacenado del archivo .config del proyecto

- En el Bloc de notas, abra el archivo .config almacenado en la carpeta del proyecto.
- Busque la entrada **EventLog1.Log** que utilizó en el procedimiento anterior.
- Cambie el valor en esa línea a **System** o **Security** y, a continuación, guarde y cierre el archivo.
- Presione F5 para ejecutar la aplicación. El nuevo valor se reflejará al hacer clic en el botón.

El archivo de configuración de la carpeta del proyecto se denomina App.config. Cuando se genera un proyecto, se copia en la carpeta de resultados y se le cambia el nombre a *NombreEnsamblado.config*, donde *NombreEnsamblado* es el resultado del proyecto; por ejemplo, MiBiblioteca.dll o MiAplicación.exe. Debe modificar esta copia del archivo .config para que afecte a una aplicación implementada. Para obtener más información, vea [Cambiar el valor de propiedades dinámicas](#).

#### Para modificar el valor almacenado del archivo .config implantado

- En el Bloc de notas, abra el archivo .config de la aplicación desde su ubicación (normalmente, la carpeta \bin u \obj\debug de la carpeta del proyecto).
- Busque la entrada **EventLog1.Log** que utilizó en el procedimiento anterior.
- Cambie el valor en esa línea a **System** o **Security** y, a continuación, guarde y cierre el archivo.
- Ejecute la aplicación desde su ubicación de generación. Ésta es la misma carpeta donde modificó el archivo .config en el paso 1. Si hace clic en el botón, se debe reflejar el nuevo valor.

#### Vea también

[Introducción a las propiedades dinámicas](#) | [Configurar aplicaciones utilizando propiedades dinámicas](#) | [Tutorial sobre propiedades dinámicas](#)

# Tutoriales sobre localización

Estos tutoriales le mostrarán el proceso de personalización de una aplicación para adaptarla a una referencia cultural o una configuración regional concretas.

## En esta sección

### [Localización de formularios Windows Forms con Visual J#](#)

Proporciona instrucciones detalladas de ejemplo sobre cómo adaptar formularios Windows Forms.

### [Localizar páginas de formularios Web Forms con Visual J#](#)

Proporciona instrucciones detalladas de ejemplo sobre cómo adaptar páginas de formularios Windows Forms.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, componentes y controles, trabajar con Windows Forms y Web Forms, y desarrollar tareas relacionadas con datos.

### [Globalizar y localizar aplicaciones](#)

Sección de Visual Basic/Visual C# que trata los pasos necesarios para lograr que una aplicación resulte útil para un público internacional.

# Tutorial: localizar formularios Windows Forms con Visual J#

El sistema de proyectos de Visual Studio proporciona una considerable compatibilidad con la localización de aplicaciones para formularios Windows Forms. Hay dos formas de generar archivos de recursos mediante el entorno de desarrollo de Visual Studio: una consiste en hacer que el sistema de proyectos genere los archivos de recursos para los elementos localizables de la interfaz de usuario, tales como el texto y las imágenes del formulario. A continuación, los archivos de recursos se incorporan a ensamblados satélite. El segundo método consiste en agregar una plantilla de archivo de recursos y, a continuación, editar la plantilla con el Diseñador XML. Una razón para hacer esto último es hacer adaptables cadenas que aparecen en cuadros de diálogo y mensajes de error. En este caso, deberá escribir código para tener acceso a estos recursos.

Este tema tutorial muestra ambos procesos en un solo proyecto de Aplicación para Windows.

También puede convertir archivos de texto a archivos de recursos; para obtener más información, vea [Recursos en formato de archivo de texto](#) y [Generador de archivos de recursos \(Resgen.exe\)](#).

## Para hacer que Visual Studio genere archivos de recursos automáticamente

1. Cree una nueva Aplicación para Windows denominada "WindowsApplication1". Para obtener información detallada, vea [Crear un proyecto de aplicación para Windows](#).
2. En la ventana Propiedades, establezca la propiedad **Localizable** del formulario en **true**.

La propiedad **Language** ya está establecida en (**Default**).

3. Arrastre al formulario un control **Button** desde la ficha Windows Forms del **Cuadro de herramientas** y establezca su propiedad **Text** en "Hello World".
4. Establezca la propiedad **Language** del formulario en "Alemán (Alemania)".
5. Establezca la propiedad **Text** del botón en "Hallo Welt".
6. Establezca la propiedad **Language** del formulario en "Francés (Francia)".
7. Establezca la propiedad **Text** del botón en "Bonjour Monde". Puede cambiar el tamaño del botón para dejar espacio suficiente para la cadena más larga, si es necesario.
8. Guarde y genere la solución.
9. Haga clic en el botón "Mostrar todos los archivos" en el Explorador de soluciones.

Los archivos de recursos aparecen bajo Form1.jsl. Form1.resx es el archivo de recursos para la referencia cultural de reserva, que se incorporará al ensamblado principal. Form1.de-DE.resx es el archivo de recursos para alemán tal como se habla en Alemania. Form1.fr-FR.resx es el archivo de recursos para francés tal como se habla en Francia.

Además, verá que los archivos aparecen con el nombre Form1.de.resx y Form1.fr.resx. Visual Studio crea estos archivos automáticamente para solucionar una limitación de Visual SourceSafe relacionada con la adición de archivos nuevos a un proyecto durante una operación Guardar. Los archivos .resx están vacíos y no contienen recursos.

10. Presione la tecla F5 o elija **Iniciar** en el menú **Depurar**.

Verá un cuadro de diálogo con un saludo en inglés, francés o alemán, según cuál sea el idioma de IU del sistema operativo.

En el procedimiento siguiente se muestra cómo establecer la referencia cultural de la IU de modo que la aplicación muestre los recursos en francés. En una aplicación real, no codificaría la referencia cultural de la IU de este modo. La configuración de la referencia cultural de la IU depende, en su lugar, de una configuración del usuario o de la aplicación.

## Para establecer la referencia cultural de IU para ver recursos específicos

1. En el Editor de código, agregue el código siguiente al principio del módulo, antes de la declaración de Form1:

```
// Visual J#
import System.Globalization.*;
import System.Threading.*;
```

2. Agregue el código siguiente: En Visual J#, debe ir en **Form1()** y también antes de la llamada a la función **InitializeComponent**.

```
// Visual J#
// Sets the UI culture to French (France)
System.Threading.Thread.get_CurrentThread().set_CurrentUICulture( new CultureInfo("fr-FR")
));
```

3. Guarde y genere la solución.
4. Presione la tecla F5 o elija **Iniciar** en el menú **Depurar**.

Ahora el formulario se mostrará en francés. Si antes cambió el tamaño del botón para dejar espacio a la cadena en francés, más larga, observe que el tamaño del botón se guardó también en el archivo de recursos en francés.

### Para agregar archivos de recursos al proyecto y editarlos de forma manual

1. En el menú **Proyecto**, haga clic en **Agregar nuevo elemento**.
2. En el cuadro Plantillas, seleccione la plantilla **Archivo de recursos de ensamblado**. Escriba el nombre de archivo "WinFormStrings.resX" en el cuadro **Nombre**. El archivo WinFormStrings.resx contendrá los recursos de reserva en inglés. Se obtendrá acceso a estos recursos cada vez que la aplicación no pueda encontrar recursos más adecuados para la referencia cultural de la interfaz de usuario.

El archivo se agregará al proyecto en el Explorador de soluciones y se abrirá automáticamente en la vista Datos del Diseñador XML.

3. En el panel **Tablas de datos**, seleccione **data**.
4. En el panel **Datos**, haga clic en una fila vacía; escriba "strMessage" en la columna **name** y "Hello World" en la columna **value**.

No es necesario especificar el valor de **type** o **mimetype** para una cadena; se utilizan para objetos. El especificador de tipo contiene el tipo de datos del objeto que se está guardando. El especificador de tipo MIME contiene el tipo base (base64) de la información binaria almacenada, si el objeto consta de datos binarios.

5. En el menú **Archivo**, haga clic en **Guardar WinFormStrings.resx**.
6. Realice los pasos 1 a 5 dos veces más para crear dos archivos de recursos más denominados "WinFormStrings.de-DE.resx" y "WinFormStrings.fr-FR.resx", con los recursos de cadena especificados en la tabla siguiente. El archivo WinFormStrings.de-DE.resx contendrá recursos que son específicos del alemán hablado en Alemania. El archivo WinFormStrings.fr-FR.resx contendrá recursos que son específicos del francés hablado en Francia.

Nombre del archivo de recursos	Name	Value
WinFormStrings.de-DE.resx	strMessage	Hallo Welt
WinFormStrings.fr-FR.resx	strMessage	Bonjour le Monde

### Para tener acceso a los recursos que se agregaron manualmente

1. En el Editor de código, importe el espacio de nombres `System.Resources` al principio del módulo de código.

```
// Visual J#
import System.Resources.*;
```

2. En la vista Diseño, haga doble clic en el botón para mostrar el código del controlador de eventos **Click** y agregue el código siguiente. El constructor **ResourceManager** toma dos argumentos. El primero es el nombre raíz de los recursos, es decir, el nombre del archivo de recursos sin la referencia cultural ni los sufijos .resx. El segundo argumento es el ensamblado principal.

En este tutorial, no se han declarado espacios de nombres, por lo tanto el primer argumento del constructor **ResourceManager** puede tomar la forma "ProjectName.ResourceFileRootName". No obstante, en una aplicación real, debería establecer la propiedad `DefaultNamespace`. En ese caso, tendrá que declarar el gestor de recursos con el nombre raíz completo del archivo de recursos, incluido el espacio de nombres. Por ejemplo, si el espacio de nombres predeterminado es "MyCompany.MyApplication.MyComponent", el primer argumento del constructor **ResourceManager** podría ser "MyCompany.MyApplication.MyComponent.WinFormsStrings".

```
// Visual J#
// Declare a Resource Manager instance
ResourceManager LocRM = new ResourceManager("WindowsApplication1.WinFormsStrings",
    System.Type.GetType("WindowsApplication1.Form1").get_Assembly());
// Assign the string for the "strMessage" key to a messagebox
MessageBox.Show(LocRM.GetString("strMessage"));
```

**Nota** De forma predeterminada, el objeto **ResourceManager** distingue entre mayúsculas y minúsculas. Si desea realizar consultas sin distinción de mayúsculas y minúsculas para que "TXTWELCOME" recupere el mismo recurso que "txtWelcome", puede establecer la propiedad [IgnoreCase](#) del gestor de recursos en **true**. No obstante, por motivos de rendimiento, siempre es preferible especificar las mayúsculas y minúsculas correctas para los nombres de recursos. Las consultas de recursos sin distinción de mayúsculas y minúsculas pueden producir problemas de rendimiento.

3. Genere y ejecute el formulario. Haga clic en el botón.

En el cuadro de mensaje, se mostrará una cadena adecuada para la configuración de referencia cultural de la interfaz de usuario; o, si no se puede encontrar un recurso para la referencia cultural de la interfaz de usuario, se mostrará una cadena de los recursos de reserva.

### **Vea también**

[Establecer los valores Culture y UI Culture para la globalización de formularios Windows Forms](#) |  
[Introducción a aplicaciones internacionales en Visual Basic y Visual C#](#) |  
[Tutorial: localizar páginas de formularios Web Forms con Visual J#](#) | [Globalizar y localizar aplicaciones](#) |  
[Tutoriales sobre localización](#)



# Tutorial: localizar páginas de formularios Web Forms con Visual J#

Cuando adapta páginas de formularios Web Forms, el sistema de proyectos de Visual Studio no genera automáticamente archivos de recursos. Es necesario crear y modificar los archivos de recursos XML. Este tema del tutorial muestra cómo agregar plantillas de archivo de recursos y luego modificar la plantilla con el Diseñador XML para crear recursos en inglés, alemán y el alemán hablado en Austria. En este tema también se describe cómo escribir código para obtener acceso a estos recursos.

También puede convertir archivos de texto a archivos de recursos. Para obtener más información, vea

[Recursos en formato de archivo de texto](#) y [Generador de archivos de recursos \(Resgen.exe\)](#).

## Para crear y editar manualmente un archivo de recursos

1. Cree una nueva aplicación Web ASP.NET denominada "LocProject". Para obtener información detallada, vea [Crear proyectos Web](#).
2. En el menú **Proyecto**, haga clic en **Agregar nuevo elemento**.
3. En el cuadro Plantillas, seleccione la plantilla **Archivo de recursos de ensamblado**. Escriba el nombre de archivo "strings.resx" en el cuadro **Nombre**. El archivo strings.resx contendrá los recursos de reserva en inglés. Se obtendrá acceso a estos recursos cada vez que la aplicación no pueda encontrar recursos más adecuados para la referencia cultural de la interfaz de usuario.

El archivo se agregará al proyecto en el Explorador de soluciones y se abrirá automáticamente en la vista Datos del Diseñador XML.

4. En el panel **Tablas de datos**, seleccione **data**.
5. En el panel **Datos**, haga clic en una fila vacía; escriba "txtSearch" en la columna **name** y "Search for the following text:" en la columna **value**.

No es necesario especificar el valor de **type** o **mimetype** para una cadena; se utilizan para objetos. El especificador de tipo contiene el tipo de datos del objeto que se está guardando. El especificador de tipo MIME contiene el tipo base (base64) de la información binaria almacenada, si el objeto consta de datos binarios.

6. Haga clic en otra fila vacía y escriba "txtWelcome" en la columna **name** y "Welcome to Localization!" en la columna **value**.
7. En el menú Archivo, haga clic en **Guardar strings.resx**
8. Realice los pasos 1 a 5 dos veces más para crear dos archivos de recursos más denominados "strings.de.resx" y "strings.de-AT.resx", que contienen los recursos de cadena especificados en la tabla siguiente. El archivo strings.de.resx contendrá recursos de alemán que son neutros, es decir, se aplican a cualquier región de habla alemana. El archivo strings.de-AT.resx contendrá recursos de alemán que son específicos de Austria.

Nombre del archivo de recursos	Name	Value
strings.de.resx	txtSearch	Nach dem folgenden Text suchen:
strings.de.resx	txtWelcome	Willkommen zur Lokalisierung!
strings.de-AT.resx	txtWelcome	Grüßt's euch zur ASP.NET Lokalisierung!

Para escribir los caracteres acentuados, puede copiar y pegar desde este tema de Ayuda, utilizar una distribución de teclado en alemán o utilizar la aplicación mapa de caracteres, charmap.exe.

## Para tener acceso a los recursos

1. En el Editor de código, importe los espacios de nombres `System.Resources` y `System.Globalization` al principio del módulo de código. Este código debe encontrarse después de la declaración **package**.

```
// Visual J#
import System.Resources.*;
import System.Globalization.*;
```

2. Agregue el código siguiente a las declaraciones que se encuentran al principio de la clase WebForm1.

```
// Visual J#
protected ResourceManager LocRM;
```

3. Agregue el código siguiente al método **Page\_Load** de la clase de código subyacente WebForm1. El constructor **ResourceManager** toma dos argumentos. El primero es el nombre raíz de los recursos, es decir, el nombre del archivo de recursos sin la referencia cultural ni los sufijos .resx. El segundo argumento es el ensamblado principal.

```
// Visual J#
LocRM= new ResourceManager("LocProject.strings", System.Type.GetType("LocProject.WebForm1").get_Assembly());
```

#### Para mostrar una cadena de recurso estático en la página

1. En vista HTML, agregue texto estático procedente de un archivo de recursos por medio del siguiente código HTML. Incluya este código entre las etiquetas <form> y </form>.

```
<h3><%=LocRM.GetString("txtWelcome")%></h3>
```

**Nota** De forma predeterminada, el objeto **ResourceManager** distingue entre mayúsculas y minúsculas. Si desea realizar consultas sin distinción de mayúsculas y minúsculas para que "TXTWELCOME" recupere el mismo recurso que "txtWelcome", puede establecer la propiedad **IgnoreCase** del gestor de recursos en true. No obstante, por motivos de rendimiento, siempre es preferible especificar las mayúsculas y minúsculas correctas para los nombres de recursos. Las consultas de recursos sin distinción de mayúsculas y minúsculas pueden producir problemas de rendimiento.

#### Para asignar una cadena de recursos a la propiedad de un control

1. En la vista Diseño, arrastre un control **Button** desde la ficha Web Forms del Cuadro de herramientas hasta el formulario.
2. En la ventana Propiedades, elimine el valor de la propiedad **Text** del botón.
3. En el Editor de código, agregue el código siguiente al método **Page\_Load** para recuperar, desde el archivo de recursos, el texto del botón.

```
// Visual J#
Button1.set_Text(LocRM.GetString("txtSearch"));
```

4. En el Editor de código, agregue el espacio de nombres `System.Threading` al principio del módulo de código.

```
// Visual J#
// Import this namespace at the beginning of the code module.
import System.Threading.*;
```

5. Agregue el código siguiente al principio del método **Page\_Load** para especificar que en la página de formularios Web Forms se deberían mostrar los recursos y el formato de acuerdo a la referencia cultural especificada en el encabezado de idioma aceptado del explorador. Este código es importante porque, de lo contrario, la aplicación Web tomará la configuración de referencia cultural predeterminada del servidor Web, que podría no resultar adecuada para el usuario remoto que consulta la página de formularios Web Forms.

```
// Visual J#
// Set the culture and UI culture to the browser's accept language
System.String[] lang =this.get_Request().get_UserLanguages();
System.Threading.Thread.get_CurrentThread().set_CurrentCulture(CultureInfo.CreateSpecificCulture(lang[0]));
System.Threading.Thread.get_CurrentThread().set_CurrentUICulture(new CultureInfo(lang[0]));
```

6. Guarde el proyecto.
7. En el menú **Generar**, elija el comando **Generar LocProject**.
8. Haga clic con el botón secundario del *mouse* (ratón) en el archivo WebForm1.aspx en el Explorador de soluciones y elija **Ver en el explorador**. Si está utilizando un sistema operativo en alemán o austriaco, verá la página en alemán o austriaco; si no es así, la verá en inglés.

Si ve la página primero en un sistema configurado para alemán y luego en un sistema configurado para austriaco,

observará que el texto de bienvenida ha cambiado a los recursos austriacos, pero el texto del botón de búsqueda no ha cambiado. Este es un ejemplo del sistema de recursos de reserva: los recursos austriacos sólo cambiaron el texto de bienvenida; el texto restante, de forma predeterminada, adopta el valor del recurso de idioma alemán neutro.

### **Vea también**

[Establecer los valores Culture y UI Culture para la globalización de formularios Web Forms](#) |

[Introducción a aplicaciones internacionales en Visual Basic y Visual C#](#) |

[Tutorial: localizar formularios Windows Forms con Visual J#](#) | [Globalizar y localizar aplicaciones](#) | [Tutoriales sobre localización](#)

# Tutoriales sobre accesibilidad

Estos tutoriales le mostrarán algunos problemas relacionados con la creación de aplicaciones accesibles.

## En esta sección

### [Crear una aplicación para Windows accesible con Visual J#](#)

Explica cómo crear una aplicación para Windows optimizada para distintos aspectos de accesibilidad.

### [Crear una aplicación Web accesible con Visual J#](#)

Explica cómo crear páginas Web accesibles para personas con discapacidades y usuarios con conexiones lentas o exploradores de texto únicamente.

## Secciones relacionadas

### [Tutoriales de Visual J#](#)

Temas paso a paso que muestran cómo crear aplicaciones Web distribuidas, componentes y controles, trabajar con Windows Forms y Web Forms, y desarrollar tareas relacionadas con datos.

### [Diseñar aplicaciones accesibles](#)

Proporciona vínculos con temas centrados en el desarrollo de aplicaciones aptas para ser utilizadas por el mayor espectro de usuarios.

# Tutorial: crear una aplicación para Windows accesible con Visual J#

Crear una aplicación accesible tiene importantes implicaciones para los negocios. Muchos gobiernos tienen normas de accesibilidad para la adquisición de software. El logotipo de certificación para Windows incluye requisitos de accesibilidad. Se calcula que sólo en Estados Unidos ya hay 30 millones de residentes, muchos de ellos clientes potenciales, se ven afectados por la accesibilidad del software.

Este tutorial trata acerca de los cinco requisitos de accesibilidad del logotipo de certificación para Windows. Según estos requisitos, una aplicación accesible:

- Admite las configuraciones de tamaño, color, fuente y entrada del Panel de control
- Admite el modo de Contraste alto
- Proporciona acceso documentado mediante teclado a todas las funciones
- Expone la ubicación del foco del teclado de forma visual y mediante programación
- Evita ofrecer información importante únicamente por medio de sonido

Para obtener más información, vea [Diseñar aplicaciones accesibles](http://msdn.microsoft.com/certification/default.asp), la página Web del programa MSDN Online Certified for Windows (<http://msdn.microsoft.com/certification/default.asp>) y la página Web de Windows XP Application Specification (<http://www.microsoft.com/windowsxp/partners/dfwspec.asp>).

## Planear la accesibilidad

Los controles del cuadro de herramientas estándar para formularios Windows Forms admiten muchas de las directrices de accesibilidad, entre ellas la exposición del foco del teclado y de los elementos de la pantalla. Puede utilizar las propiedades de los controles para ofrecer compatibilidad con otras directrices de accesibilidad, como se muestra en la tabla siguiente. Además, es recomendable utilizar menús para proporcionar acceso a las funciones del programa.

Propiedad de control	Consideraciones para la accesibilidad
AccessibleDescription	Descripción que se ofrece a las ayudas de accesibilidad, tales como los lectores de pantalla. Las ayudas de accesibilidad son programas y dispositivos especializados, que ayudan a las personas con discapacidades a utilizar de forma más eficiente los equipos.
AccessibleName	Nombre que se ofrece a las ayudas de accesibilidad.
AccessibleRole	Describe el uso del elemento en la interfaz de usuario.
TabIndex	Crea una ruta de desplazamiento razonable a través del formulario. Es importante que los controles que no tienen etiquetas intrínsecas, tales como los cuadros de texto, vayan precedidos de forma inmediata en el orden de tabulación por sus etiquetas asociadas.
Text	Utilice el carácter "&" para crear teclas de acceso. El uso de teclas de acceso forma parte de la tarea de proporcionar acceso documentado mediante teclado a las funciones.
FontSize	Si no es posible ajustar el tamaño de la fuente, debe establecerse en 10 puntos o más. Una vez establecido el tamaño de la fuente del formulario, todos los controles que se agreguen a continuación al formulario tendrán el mismo tamaño.
ForeColor	Si se establece esta propiedad en el valor predeterminado, el formulario utilizará las preferencias de color del usuario.
BackColor	Si se establece esta propiedad en el valor predeterminado, el formulario utilizará las preferencias de color del usuario.
BackgroundImage	Deje esta propiedad en blanco para que el texto resulte más legible.

Para obtener información acerca de la compatibilidad con diferentes diseños de teclado, vea [Planear aplicaciones de uso internacional](#).

## Crear el proyecto

Este tutorial crea la interfaz de usuario para una aplicación que admite pedidos de pizzas. Consta de un control **TextBox** para el nombre del cliente, un grupo **RadioButton** para seleccionar el tamaño de la pizza, un control **CheckedListBox** para seleccionar los ingredientes, dos controles **Button** con las etiquetas Order (Pedir) y Cancel (Cancelar) y un **Menú** con un comando Exit (Salir).

El usuario escribe el nombre del cliente, el tamaño de la pizza y los ingredientes que desea. Cuando el usuario hace clic en el botón Order, un cuadro de mensaje muestra un resumen del pedido y su precio; los controles se borran y se preparan para el siguiente pedido. Cuando el usuario hace clic en el botón Cancel, los controles se borran y se preparan para el siguiente pedido.

Cuando el usuario hace clic en el elemento de menú Exit, el programa se cierra.

El énfasis de este tutorial no se encuentra en el código del sistema de pedidos al detalle, sino en la accesibilidad de la interfaz de usuario. El tutorial muestra las funciones de accesibilidad de varios controles de uso frecuente, entre ellos botones, botones de opción, cuadros de texto y etiquetas.

## Para comenzar a crear la aplicación

- Cree una aplicación para Windows nueva en Visual J#. Asigne al proyecto el nombre **PedidoPizza**. (Para obtener información detallada, vea [Crear nuevas soluciones y proyectos](#).)

## Agregar los controles al formulario

Cuando agregue los controles a un formulario, tenga en cuenta las siguientes directrices para crear una aplicación accesible:

- Establezca las propiedades **AccessibleDescription** y **AccessibleName**. En este ejemplo, es suficiente la configuración **Default** para **AccessibilityRole**. Para obtener más información acerca de las propiedades de accesibilidad, vea [Proporcionar información sobre la accesibilidad de los controles en un formulario Windows Form](#).
- Establezca el tamaño de fuente en 10 puntos o un tamaño mayor.

**Nota** Si al principio establece el tamaño de fuente del formulario en 10, todos los controles que agregue después al formulario tendrán un tamaño de fuente de 10.

- Compruebe que los controles **Label** que describen controles **TextBox** precedan inmediatamente al control **TextBox** correspondiente en el orden de tabulación.
- Agregue una tecla de acceso, mediante el carácter "&", a la propiedad **Text** de los controles a los que el usuario pueda desear desplazarse.
- Agregue una tecla de acceso, mediante el carácter "&", a la propiedad **Text** de las etiquetas que precedan un control al que el usuario pueda desear desplazarse. Establezca la propiedad **UseMnemonic** de las etiquetas como **true**, de modo que el foco se establezca en el siguiente control del orden de tabulación, cuando el usuario presione la tecla de acceso.
- Agregue teclas de acceso a todos los elementos de menú.

## Para hacer accesible la aplicación para Windows

- Agregue los controles al formulario y establezca las propiedades tal como se describe a continuación. Vea la imagen que aparece al final de la tabla para ver un modelo de la disposición de los controles en el formulario. Para obtener más información sobre cómo trabajar con el control **MainMenu**, vea [Agregar menús y elementos de menú a formularios Windows Forms](#).

Objeto	Propiedad	Valor
<b>Form1</b>	AccessibleDescription	Order form
	AccessibleName	Order form
	FontSize	10
	Text	Pizza Order Form
<b>PictureBox</b>	Name	logo
	AccessibleDescription	A slice of pizza
	AccessibleName	Company logo
	Imagen	Cualquier icono o mapa de bits
<b>Label</b>	Name	companyLabel
	Text	Good Pizza
	TabIndex	1
	AccessibleDescription	Company name
	AccessibleName	Company name
	BackColor	Blue
	ForeColor	Yellow
<b>Label</b>	FontSize	18
	Name	customerLabel
	Text	&Name
	TabIndex	2
	AccessibleDescription	Customer name label
	AccessibleName	Customer name label

	UseMnemonic	True
<b>TextBox</b>	Name	customerName
	Text	(ninguno)
	TabIndex	3
	AccessibleDescription	Customer name
	AccessibleName	Customer name
<b>GroupBox</b>	Name	sizeOptions
	AccessibleDescription	Pizza size options
	AccessibleName	Pizza size options
	Text	Pizza size
	TabIndex	4
<b>RadioButton</b>	Name	smallPizza
	Text	&Small \$6.00
	Checked	True
	TabIndex	0
	AccessibleDescription	Small pizza
	AccessibleName	Small pizza
<b>RadioButton</b>	Name	largePizza
	Text	&Large \$10.00
	TabIndex	1
	AccessibleDescription	Large pizza
	AccessibleName	Large pizza
<b>Label</b>	Name	toppingsLabel
	Text	&Toppings (\$0.75 each)
	TabIndex	5
	AccessibleDescription	Toppings label
	AccessibleName	Toppings label
	UseMnemonic	True
<b>CheckedListBox</b>	Name	toppings
	TabIndex	6
	AccessibleDescription	Available toppings
	AccessibleName	Available toppings
	Items	Pepperoni, Sausage, Mushrooms
<b>Button</b>	Name	order
	Text	&Order
	TabIndex	7
	AccessibleDescription	Total the order
	AccessibleName	Total order
<b>Button</b>	Name	cancel
	Text	&Cancel
	TabIndex	8
	AccessibleDescription	Cancel the order
	AccessibleName	Cancel order
<b>MainMenu</b>	Name	theMainMenu
<b>MenuItem</b>	Name	fileCommands
	Text	&File
<b>MenuItem</b>	Name	exitApp
	Text	E&xit

El formulario tendrá una apariencia parecida a la siguiente:

### Para agregar funcionalidad al formulario y a sus controles

1. Agregue el código siguiente al procedimiento de evento **Click** del botón Pedido, para calcular e informar del costo total y borrar los controles.

```
// Visual J#
private void ResetOrder()
{
    customerName.setText("");
    smallPizza.set_Checked(true);
    for (int topping = 0; topping < toppings.getItems().get_Count(); topping++)
    {
        toppings.SetItemChecked(topping, false);
    }
}

private void order_Click (Object sender, System.EventArgs e)
{
    final System.Decimal ToppingCost =new System.Decimal(0.75);
    final System.Decimal SmallCost = new System.Decimal (6);
    final System.Decimal LargeCost =new System.Decimal(10);

    System.Decimal cost=new System.Decimal(0);
    cost = System.Decimal.Add((smallPizza.get_Checked() ? SmallCost : LargeCost),
    System.Decimal.Multiply((new System.Decimal(toppings.get_CheckedIndices().get_Count())
    ), ToppingCost));
    System.String orderMessage;
    orderMessage = System.String.Format("Total cost of pizza for {0} is {1:C}.", custome
    rName.get_Text(), cost);
    MessageBox.Show(orderMessage);
    ResetOrder();
}
```

2. Agregue el código siguiente al procedimiento de evento **Click** para que el botón Cancelar borre los controles.

```
// Visual J#
private void cancel_Click (Object sender, System.EventArgs e)
{
    ResetOrder();
}
```

3. Haga clic en el control **MainMenu** del formulario y, a continuación, haga doble clic en **Exit MenuItem** para crear su método de evento **Click**. Agregue el código siguiente al procedimiento de evento **Click** para que el elemento de menú **Salir** cierre la aplicación.

```
// Visual J#
```



```
private void exitApp_Click (Object sender, System.EventArgs e)
{
    this.Dispose();
}
```

## Compatibilidad con el modo de contraste alto

El modo de Contraste alto es una configuración del sistema de Windows que mejora la legibilidad; para ello, utiliza colores contrastados y tamaños de fuente de fácil lectura para los usuarios con discapacidad visual. La propiedad [SystemInformation.HighContrast](#) se proporciona para determinar si está establecido el modo de contraste alto.

Si **SystemInformation.HighContrast** es **true**, la aplicación debe:

- Mostrar todos los elementos de la interfaz de usuario con la combinación de colores del sistema
- Ofrecer mediante indicaciones visuales o sonido cualquier información que se ofrezca mediante el color. Por ejemplo, si determinados elementos de una lista se resaltan mediante una fuente de color rojo, podría agregar a la fuente el formato de negrita, de modo que el usuario disponga de una indicación, distinta del color, de que los elementos están resaltados.
- Omitir las imágenes o tramas que se encuentren detrás del texto

La aplicación debe comprobar la configuración de **HighContrast** cuando se inicie, así como responder al evento de sistema **UserPreferenceChanged**. El evento **UserPreferenceChanged** se provoca siempre que cambia el valor de **HighContrast**.

En nuestra aplicación, el único elemento que no utiliza las configuraciones del sistema para el color es `lblCompanyName`. La [clase SystemColors](#) se utiliza para cambiar las configuraciones de color de la etiqueta a los colores del sistema seleccionados por el usuario.

### Para habilitar el modo de Contraste alto de forma eficaz

1. Cree un método para establecer los colores de la etiqueta en los colores del sistema.

```
// Visual J#
private void SetColorScheme()
{
    if (SystemInformation.get_HighContrast())
    {
        companyName.set_BackColor(SystemColors.get_Window());
        companyName.set_ForeColor(SystemColors.get_WindowText());
    }
    else
    {
        companyName.set_BackColor(Color.get_Blue());
        companyName.set_ForeColor(Color.get_Yellow());
    }
}
```

2. Llame al procedimiento `SetColorScheme` del constructor de formularios (`public Form1` en Visual J#).

```
// Visual J#
public Form1()
{
    InitializeComponent();
    SetColorScheme();
}
```

3. Cree un procedimiento de evento, con la firma adecuada, que responda al evento **UserPreferenceChanged**.

```
// Visual J#
public void UserPreferenceChanged(Object sender, Microsoft.Win32.UserPreferenceChangeEventArgs e)
{
    SetColorScheme();
}
```

```
}
```

4. Agregue código al constructor del formulario, después de la llamada a `InitializeComponents`, para enlazar el procedimiento de evento al evento del sistema. Este método llama al procedimiento `SetColorScheme`.

```
// Visual J#
public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    SetColorScheme();
    Microsoft.Win32.SystemEvents.add_UserPreferenceChanged(
        new Microsoft.Win32.UserPreferenceChangedEventHandler(this.UserPreferenceChanged))
;
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
}
```

5. Para liberar el evento cuando se cierre la aplicación, agregue código al método **Dispose** del formulario antes de llamar al método **Dispose** de la clase base.

**Nota** El código del evento del sistema ejecuta un subproceso separado de la aplicación principal. Si no libera el evento, el código enlazado continuará ejecutándose incluso después de que se cierre el programa.

```
// Visual J#
protected void Dispose(boolean disposing)
{
    if (disposing)
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    Microsoft.Win32.SystemEvents.remove_UserPreferenceChanged(
        new Microsoft.Win32.UserPreferenceChangedEventHandler( this.UserPreferenceChange
d));
    super.Dispose(disposing);
}
```

6. Presione F5 para ejecutar la aplicación.

## Compatibilidad con las configuraciones de tamaño y fuente del Panel de control

La barra de menú, la barra de título, los bordes y la barra de estado cambian todos automáticamente de tamaño cuando el usuario cambia la configuración del Panel de control. En esta aplicación no es necesario hacer más cambios en los controles ni en el código.

## Ofrecer información importante por medios diferentes del sonido

En esta aplicación no se ofrece ninguna información exclusivamente mediante sonido. Si utiliza sonido en la aplicación, es recomendable que suministre la información también por otros medios. Algunas sugerencias sobre métodos:

- Haga parpadear la barra de título mediante la función de la API de Windows **FlashWindow**. Para obtener un ejemplo de cómo llamar a las funciones de la API de Windows, vea [Tutorial: Llamar a las API de Windows](#).

**Nota** Es posible que el usuario haya habilitado el servicio SoundSentry de Windows, que también hace que la

ventana parpadee cuando se reproducen sonidos a través del altavoz integrado en el equipo.

- Muestre la información importante en una ventana no modal, para que el usuario pueda responder a ella. (Para obtener información detallada, vea [Mostrar formularios Windows Forms modales y no modales](#).)
- Muestre un cuadro de mensaje que adquiera el foco del teclado. Evite utilizar este método cuando exista la posibilidad de que el usuario esté escribiendo.
- Muestre un indicador de estado en el área de notificación de estado de la barra de tareas. Para obtener información detallada, vea [Agregar iconos de aplicación a la barra de tareas con el control NotifyIcon de formularios Windows Forms](#).

## Probar la aplicación

Antes de implementar la aplicación, es recomendable probar las funciones de accesibilidad implementadas.

### Para probar las funciones de accesibilidad

1. Para probar el acceso mediante teclado, desconecte el mouse (ratón) y desplácese por la interfaz de usuario a cada función utilizando sólo el teclado. Asegúrese de que sea posible ejecutar todas las tareas mediante el uso exclusivo del teclado.
2. Para probar la compatibilidad con el modo de contraste alto, seleccione **Opciones de accesibilidad** en **Panel de control**. Haga clic en la ficha **Mostrar** y seleccione la casilla de verificación **Usar Contraste alto**. Desplácese por todos los elementos de la interfaz de usuario para asegurarse de que se reflejan los cambios de color y fuente. Asegúrese también de que se omiten las imágenes y tramas que se representan como fondo del texto.

**Nota** Windows NT 4 no dispone del icono **Opciones de accesibilidad** en **Panel de control**. Por ello, este procedimiento para cambiar el valor **SystemInformation.HighContrast** no funciona en Windows NT 4.

3. Para comprobar la exposición del foco de teclado, ejecute el Ampliador. (Para abrirlo, Haga clic en el menú **Inicio**, seleccione **Programas, Accesorios, Accesibilidad** y haga clic en **Ampliador**). Desplácese por la interfaz de usuario utilizando tanto la tabulación de teclado como el mouse. Asegúrese de que el Ampliador sigue correctamente todos los desplazamientos.
4. Para probar la exposición de los elementos de la pantalla, ejecute Inspect y utilice tanto el *mouse* (ratón) como la tecla **TAB** para desplazarse hasta cada elemento. Asegúrese de que la información que se presenta en los campos Nombre, Estado, Función, Ubicación y Valor de la ventana Inspect tiene sentido para el usuario, para cada uno de los objetos de la interfaz de usuario. Inspect se instala como parte de Microsoft® Active Accessibility® SDK, que se encuentra disponible en <http://www.microsoft.com/enable/msaa/>.

### Vea también

[Instrucciones de diseño del software para la accesibilidad](#) | [Tutoriales sobre accesibilidad](#)

# Tutorial: crear una aplicación Web accesible con Visual J#

La creación de páginas Web accesibles permite llegar al máximo número posible de usuarios. Las personas con discapacidades no son los únicos usuarios que apreciarán que las páginas sean accesibles. Los usuarios que tengan conexiones lentas o exploradores que sólo muestren texto aprovecharán también las opciones de accesibilidad. El diseño accesible permite que herramientas de automatización tales como los motores de búsqueda busquen, indiquen y manipulen la información de las páginas. Por último, es posible que la legislación futura sobre telecomunicaciones requiera que la información que se distribuya por Internet resulte accesible de la misma manera que el software tradicional.

Para obtener más información, vea [Instrucciones de diseño Web para la accesibilidad](#).

A continuación se muestran, procedentes del sitio Web de accesibilidad de Microsoft, las "12 Sugerencias para el diseño de páginas Web accesibles":

- Utilice texto ALT adecuado para todos los gráficos.
- Utilice correctamente los mapas de imágenes.
- Utilice textos de vínculo útiles.
- Implante un buen desplazamiento por el teclado.
- Proporcione alternativas para todos los controles y subprogramas.
- Páginas alternativas que no utilicen marcos.
- Utilice correctamente las tablas y sus alternativas.
- Proporcione compatibilidad con las opciones de formato del lector.
- No exija la utilización de hojas de estilos.
- Utilice formatos de archivo que el lector pueda utilizar.
- Evite utilizar marquesinas que se desplacen.
- Proporcione títulos para la mayoría de los objetos.

Si no puede cumplir los objetivos de accesibilidad, considere la posibilidad de proporcionar páginas Web alternativas que sólo contengan texto.

## Planear la accesibilidad

Los controles del cuadro de herramientas estándar para formularios Web Forms admiten muchas de las directrices de accesibilidad, entre ellas la exposición del foco del teclado y de los elementos de la pantalla. Puede utilizar las propiedades de los controles para ofrecer compatibilidad con otras directrices de accesibilidad, como se muestra en la tabla siguiente.

Propiedad de control	Consideraciones de accesibilidad
TabIndex	Utilice la propiedad <b>TabIndex</b> para crear una ruta de desplazamiento razonable a través del formulario. Es importante que los controles que no tienen etiquetas intrínsecas, tales como los cuadros de texto, vayan precedidos de forma inmediata en el orden de tabulación por sus etiquetas asociadas.
Text	Utilice la etiqueta HTML <code>&lt;u&gt;</code> para mostrar la tecla de acceso de los controles. El uso de teclas de acceso contribuye a proporcionar acceso documentado mediante teclado a todas las funciones. (Utilice la propiedad <b>AccessKey</b> para habilitar teclas de acceso para los controles).
FontSize	Utilice etiquetas de encabezado en lugar de tamaños específicos.
AlternateText	Proporcione texto alternativo para todas las imágenes.
AccessKey	Utilice esta propiedad para proporcionar acceso a los controles mediante el teclado.

## Crear el proyecto

Este tutorial crea un sitio Web para una clase sobre álgebra. La interfaz de usuario consta de un control **Image** para el logotipo de la clase, un control **Panel** con varios controles **HyperLink** para las lecciones, un control **DataGrid** para mostrar una tabla con el plan de estudios y un control **HyperLink** que enlaza con una página que describe el libro de texto para las clases.

El objetivo de este tutorial es la accesibilidad de la interfaz de usuario y cómo codificar los controles de servidor Web para que permitan la accesibilidad. Muestra las funciones de accesibilidad de varios controles de uso frecuente, entre ellos los controles **Link**, **DataGrid**, **Image** y **Label**. Todos los controles **HyperLink** conducen a la misma página Web, que estará en blanco.

### Para comenzar a crear la aplicación

1. Cree una aplicación Web ASP.NET nueva en Visual J#. Asigne al proyecto el nombre **ClaseDeÁlgebra**. (Para obtener

información detallada, vea [Crear nuevas soluciones y proyectos.](#))

El formulario de inicio, WebForm1.aspx, aparecerá en el Diseñador de Web Forms.

- En el menú **Proyecto**, haga clic en **Agregar formulario Web**. Aparecerá el cuadro de diálogo **Agregar nuevo elemento**. Asigne al nuevo formulario Web Forms el nombre Lecture.aspx y haga clic en **Abrir**. Este formulario se utilizará para proporcionar la dirección **NavigationUrl** a los ejemplos de **HyperLink** de WebForm1.
- Cierre el formulario denominado Lecture.aspx y guarde los cambios.

## Agregar los controles al formulario

Cuando agregue los controles al formulario de esta aplicación, tenga en cuenta las siguientes directrices para crear una aplicación accesible:

- Todas las imágenes deben tener texto ALT.
- Cuando use tablas, utilice el atributo TITLE para proporcionar nombres a las columnas y las filas de las tablas. Además, asegúrese de que las tablas tengan sentido si se leen de izquierda a derecha y de arriba abajo.
- Utilice etiquetas de encabezado reales, como por ejemplo H1, H2, etc., en lugar de usar texto con formato, para permitir la utilización de las opciones de formato seleccionadas por el usuario.
- Utilice texto útil en los vínculos. Por ejemplo, si el texto es "Haga clic aquí para ver las notas de la Lección 1", "Notas de la lección 1" resultará más útil que "Haga clic aquí".
- Proporcione una ruta razonable para desplazarse por la página, que siga el flujo normal de texto relativo al idioma dado.
- Utilice el atributo TITLE para la mayoría de los objetos.

## Para agregar controles accesibles a la página

- En la ventana Propiedades, seleccione **Documento** y establezca la propiedad **title** como "Clase de álgebra", la propiedad **bgColor** en #ffdab9 (color albaricoque claro) y la propiedad **pageLayout** en **GridLayout**.
- Agregue los controles al formulario y establezca las propiedades tal como se describe a continuación. Vea la imagen que aparece al final de la tabla para ver un modelo de la disposición de los controles en el formulario.

Para colocar correctamente los controles **HyperLink** del control **Panel**, deberá seleccionar el control **Panel** y colocar el puntero en **Panel**. El control **HyperLink** se agregará en la posición del puntero.

Objeto	Propiedad	Valor
Image	AlternateText	Ecuaciones algebraicas.
	ImageUrl	Cualquier mapa de bits
	TabIndex	0
Label	ID	ClassName
	ForeColor	Blue
	Text	Clase de repaso de álgebra
	FontSize	Extra grande
Panel	TabIndex	1
	ID	Lectures
	TabIndex	2
HyperLink (agregar al panel Lecciones)	ID	Lecture1
	Text	Lección 1
	AccessKey	1
	NavigateUrl	Lecture.aspx
	TabIndex	3
HyperLink (agregar al panel Lecciones)	ID	Lecture2
	Text	Lección 2
	AccessKey	2
	NavigateUrl	Lecture.aspx
	TabIndex	4
HyperLink (agregar al panel Lecciones)	ID	Lecture3

	Text	Lección 3
	AccessKey	3
	NavigateUrl	Lecture.aspx
	TabIndex	5
Label	ID	TextbookLabel
	Text	Texto:
	TabIndex	6
Hyperlink	ID	TextbookLink
	Text	Algebra Review, J. A. Smith
	NavigateUrl	Lecture.aspx
	TabIndex	7
Label	ID	SyllabusTitle
	Text	Plan de estudios
	TabIndex	8
DataGrid	ID	SyllabusGrid
	TabIndex	9

3. En el diseñador, seleccione el control **Panel**. En el diseñador, cambie el texto del panel a **Notas de las lecciones**.
4. En el diseñador, haga clic en la ficha HTML. Modifique la ficha HTML para que indique que el lenguaje de la página sea español.

```
<HTML lang="en">
```

El formulario tendrá una apariencia similar a la siguiente en tiempo de ejecución:



### Para agregar funcionalidad al formulario y a sus controles

1. Agregue la función siguiente para crear datos para el control **DataGrid**. En la mayoría de las aplicaciones, recuperaría los datos de una base de datos, pero para esta aplicación es suficiente el método siguiente.

```
// Visual J#
protected DataTable CreateSyllabusEntries()
{
    DataTable syllabus = new DataTable();
    syllabus.get_Columns().Add("Meeting Date", System.Type.GetType("System.String"));
    syllabus.get_Columns().Add("Topic", System.Type.GetType("System.String"));
    syllabus.get_Rows().Add(new Object[] { "4/2/2001",
                                           "Integers and Rational Numbers" } );
}
```

```

syllabus.get_Rows().Add(new Object[] { "4/6/2001",
                                     "Equations and Polynomials" } );
syllabus.get_Rows().Add(new Object[] { "4/8/2001",
                                     "Roots and Irrational Numbers" } );

return syllabus;
}

```

2. Modifique el método `Page_Load` para enlazar los datos del plan de estudios al control **DataGrid**.

```

// Visual J#
private void Page_Load(Object sender, System.EventArgs e)
{
    DataTable syllabus = CreateSyllabusEntries();
    SyllabusGrid.set_DataSource(syllabus);
    SyllabusGrid.DataBind();
}

```

3. Utilice el evento **ItemCreated** del control DataGrid para agregar atributos TITLE a las celdas de la tabla creadas por el control DataGrid. Como mínimo, debe establecer el atributo TITLE de los encabezados de columnas. Es probable que desee establecer también el atributo TITLE de las columnas. Para obtener más información sobre los controladores de eventos, vea [Crear controladores de eventos en páginas de formularios Web Forms](#).

```

// Visual J#
private void SyllabusGrid_ItemCreated(Object sender, System.Web.UI.WebControls.DataGridItemEventArgs e)
{
    TableCellCollection cells = e.get_Item().get_Cells();
    if (e.get_Item().get_ItemType() == ListItemType.Header)
    {
        // Add TITLE attributes to the column headers.
        cells.get_Item(0).get_Attributes().set_Item("TITLE", "Date");
        cells.get_Item(1).get_Attributes().set_Item("TITLE", "Topic");
    }
    else if ((e.get_Item().get_ItemType() == ListItemType.Item) ||
             (e.get_Item().get_ItemType() == ListItemType.AlternatingItem))
    {
        // Add descriptive titles to individual cells.
        DataRowView rowView = (DataRowView) e.get_Item().get_DataItem();
        cells.get_Item(0).get_Attributes().set_Item("TITLE", rowView.get_Row().get_ItemArray().toString());
        cells.get_Item(1).get_Attributes().set_Item("TITLE", rowView.get_Row().get_ItemArray().toString());
    }
}

```

4. Ejecute la aplicación.

## Probar la aplicación

Utilice la lista de comprobación siguiente para probar la accesibilidad de la aplicación.

- Pruebe las teclas de acceso. Para esta aplicación, ha definido las teclas de acceso ALT+1, ALT+2 y ALT+3 para los controles **HyperLink**. Las teclas de acceso lo trasladarán a los vínculos. Presione ENTRAR para seguir los vínculos.
- Desactive los gráficos para asegurarse de que el texto ALT que se muestra permite usar la página. En Internet Explorer versión 6, haga clic en el menú Herramientas y elija el comando Opciones de Internet. En la ficha Avanzadas, desactive la opción de gráficos de la sección Multimedia.
- Desactive los sonidos para comprobar que no se pierde ninguna instrucción importante. En Internet Explorer, versión 6, haga clic en el menú Herramientas y elija el comando Opciones de Internet. En la ficha Avanzadas, desactive la opción de sonido de la sección Multimedia.

- Vea la aplicación en un explorador que permita desactivar las hojas de estilos. Desactive las hojas de estilos para asegurarse de que la página continúa siendo legible.
- Utilice la opción de Contraste alto y asegúrese de que la página continúa siendo legible.
- Utilice tamaños de fuente personalizados en la sección Pantalla del Panel de control.
- Utilice la fuente de mayor tamaño permitido (disponible sólo cuando se selecciona Contraste alto) para asegurarse de que la página continúa siendo legible.
- Cambie el tamaño de la ventana del explorador y compruebe la legibilidad.
- Desplácese mediante el teclado para asegurarse de que el orden de desplazamiento es razonable, de que la tecla TAB pasa por todos los vínculos y de que CONTROL+TAB se desplaza entre paneles o secciones.
- Seleccione todo el texto y cópielo en el portapapeles, para asegurarse de que tiene sentido.
- Consulte el sitio Web Bobby para ver una herramienta de análisis que puede utilizar para examinar las páginas (<http://www.cast.org/bobby/>).

## **Vea también**

[Instrucciones de diseño del Web para la accesibilidad](#) | [Tutoriales sobre accesibilidad](#)



# Actualizar proyectos de Visual J++ 6.0

En esta sección se proporcionan recomendaciones y procedimientos para actualizar aplicaciones de Microsoft® Visual J++ 6.0® a Microsoft Visual J#. Incluye temas sobre planeamiento, solución de problemas, uso de recursos, actualización de proyectos de interoperabilidad de Java y COM, y actualización de varios paquetes com.ms.\*. También ofrece temas sobre el uso del Asistente para actualización a Visual J# y el informe de actualización.

Puede abrir y actualizar proyectos existentes de Visual J++ 6.0 utilizando Visual J#, pero, en algunos casos, puede ser necesario modificar estos proyectos después de actualizarlos.

## En esta sección

### [Introducción a Visual J# para usuarios de Visual J++](#)

Explica cómo Visual J# agrega compatibilidad con Java en .NET Framework.

### [Planear la actualización](#)

Proporciona información general sobre la actualización de Visual J++ a Visual J#.

### [Asistente para actualización a Visual J#](#)

Proporciona información general sobre el Asistente para actualización a Visual J# e información detallada sobre el trabajo con sus características.

### [Informe de actualización de Visual J#](#)

Describe el formato y el propósito del informe de actualización.

### [Actualizar proyectos de Visual J++ 6.0](#)

Explica cómo actualizar proyectos de Visual J++ 6.0 a Visual J# utilizando el Asistente para actualización.

### [Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#)

Trata el procedimiento para convertir los recursos que utiliza un proyecto de Visual J++ 6.0 en recursos de .NET Framework, incluidos los problemas de compatibilidad que supone la incrustación o vinculación de recursos.

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Explica cómo compilar, depurar y ejecutar componentes de Java o COM desarrollados con Visual J++® 6.0.

### [Actualizar aplicaciones que cargan clases dinámicamente](#)

Contiene secciones sobre la variable CLASSPATH y el proceso de enlace.

### [Actualizar aplicaciones que utilizan cargadores de clases personalizados](#)

Explica por qué no se recomienda utilizar cargadores de clases personalizados en Visual J#.

### [Actualizar aplicaciones que utilizan la semántica de seguridad de Visual J++ 6.0](#)

Trata la semántica de seguridad y las aplicaciones que utilizan proveedores de otros fabricantes.

### [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework](#)

Muestra las bibliotecas de clases de .NET Framework equivalentes a algunos de los paquetes com.ms.\* más utilizados. Incluye temas sobre la actualización de paquetes com.ms.xml.\*, com.ms.mtx y com.ms.wfc.html.

### [Problemas comunes al actualizar proyectos de Visual J++ 6.0](#)

Enumera los problemas que se pueden encontrar después de actualizar un proyecto a Visual J#.

### [Trabajar con Visual J++ 6.0 y Visual J#](#)

Explica cómo se puede trabajar con Visual J++ 6.0 y Visual J# conjuntamente después de actualizar una solución de Visual J++ 6.0.

## Secciones relacionadas

### [Compatibilidad con el lenguaje](#)

Proporciona información de referencia sobre las características de Visual J++ 6.0 y .NET Framework que son compatibles y no compatibles con Visual J#.

### [Compatibilidad con bibliotecas de clases](#)

Enumera los paquetes compatibles y no compatibles, así como información específica de implementación para la compatibilidad con bibliotecas de clases.

### [Referencia para la actualización a Visual J#](#)

Proporciona ayuda para problemas detectados por el Asistente para actualización durante la actualización de proyectos de Visual J++ 6.0 a Visual J#.

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Trata las consideraciones que se deben tener en cuenta a la hora de actualizar proyectos de Visual J++ 6.0 que utilicen la interoperabilidad de Java y COM.

### [Proyectos de Visual J#](#)

Proporciona información sobre las propiedades de los proyectos de Visual J# y cómo tener acceso a ellas.

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

# Introducción a Visual J# para usuarios de Visual J++

Visual J#:

- Agrega compatibilidad con Java en .NET Framework. Este producto ofrece la capacidad y la riqueza de .NET Framework a los programadores de Java y permite continuar utilizando conocimientos y código de Java existentes.
- Permite participar en la visión de los servicios Web XML y aprovechar los completos marcos de programación de las bibliotecas de ASP.NET, ADO.NET y Windows Forms para crear fabulosas aplicaciones.
- Proporciona una ruta de actualización sin problemas para desarrolladores de Visual J++. Está totalmente integrado con Visual Studio .NET y presenta una interfaz familiar para los desarrolladores de Visual J++.

Vea [Información general sobre Visual J#](#) para obtener una introducción al producto.

Visual J# puede coexistir con Visual J++ en el mismo sistema. Durante la actualización, no se modifican los archivos de código fuente ni otros archivos del proyecto. Tampoco se toca el archivo de proyecto existente. Puede abrir el proyecto de Visual J++ y trabajar en la aplicación en Visual J++, incluso después de haberlo actualizado a Visual J#.

A continuación, se exponen algunos aspectos clave de la actualización a Visual J#:

- El desarrollo en Visual J# está orientado a .NET Framework, es decir, el resultado de los proyectos de Visual J# son archivos binarios en el Lenguaje intermedio de Microsoft (MSIL). Estos archivos binarios se ejecutan en Common Language Runtime (CLR) con la compatibilidad de tiempo de ejecución proporcionada por la implementación de Visual J#.
- La mayor parte de la funcionalidad de las bibliotecas de JDK 1.1.4 está disponible en la implementación de Visual J#. Para obtener información detallada, vea [Compatibilidad con bibliotecas de clases](#).
- El compilador de Visual J# admite todas las características clave del lenguaje y es también totalmente compatible con el uso de .NET Framework. Asimismo, admite las extensiones compatibles con Visual J++, como J/Direct y la interoperabilidad de Java y COM.
- No ofrece compatibilidad en tiempo de diseño para proyectos WFC. Sólo las aplicaciones nuevas creadas con cualquiera de las plantillas de proyecto de Visual J# disponen de esta compatibilidad en tiempo de diseño. Sin embargo, puede ampliar aplicaciones existentes agregando funcionalidad nueva habilitada por Visual J#. Por ejemplo, puede ampliar una aplicación WFC con Windows Forms. Vea [WFCUsingWinForm \(Ejemplo\)](#) para obtener más información.
- Los formatos de archivo de recursos son diferentes. Para obtener más información, vea [Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#)
- Se puede utilizar Visual J# para abrir y actualizar proyectos creados con Visual J++ 6.0. Al abrir un proyecto de Visual J++ en Visual J#, se inicia el Asistente para actualización, que facilita la transición al nuevo producto. Este asistente crea un proyecto nuevo al que se agregan los archivos de proyecto existentes. Durante la actualización, no se modifican los archivos de código fuente. Además, el asistente analiza el proyecto para identificar posibles problemas. Estos problemas se enumeran en un informe de actualización que se agrega al proyecto nuevo. Cada problema tiene un vínculo a un tema en la documentación que explica cómo solucionarlo.

**Vea también**

[Actualizar proyectos de Visual J++ 6.0](#)

# Planear la actualización

En la mayoría de los casos, Visual J# proporciona una experiencia de actualización sencilla para proyectos de Visual J++ . Sin embargo, debe asegurarse de que el proyecto se genera y ejecuta correctamente en Visual J++ antes de intentar actualizarlo. También debe definir la configuración activa del proyecto de Visual J++ como Lanzamiento y comprobar que la aplicación se ejecuta antes de actualizarla. Esto facilita la actualización.

Hay algunas características en Visual J++ que tienen problemas de compatibilidad con Visual J#. Vea [Compatibilidad con bibliotecas de clases](#) y [Compatibilidad con el lenguaje](#) para determinar las posibles tareas necesarias para que la aplicación se ejecute correctamente en Visual J#. [Problemas comunes al actualizar proyectos de Visual J++ 6.0](#) enumera una serie de problemas que puede encontrar al actualizar sus aplicaciones.

El Asistente para actualización define las propiedades del proyecto y hace referencia a componentes COM y .NET basándose en la información que usted proporcione, así como en el análisis de los archivos de proyecto. Para proyectos de gran tamaño, el análisis puede tardar un cierto tiempo, comparable al tiempo necesario para generar el proyecto actualizado. Si bien el asistente permite omitir el análisis haciendo clic en el botón Finalizar, se recomienda no hacerlo. El informe de actualización que crea el asistente identifica muchos de los problemas para lograr que la aplicación se ejecute correctamente en Visual J#. La lista de problemas puede resultar muy útil para determinar la cantidad de trabajo necesario para una correcta actualización.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#)

# Asistente para actualización a Visual J#

Visual J# incluye un Asistente para actualización que facilita el proceso de actualización de proyectos de Visual J++ 6.0 a Visual J#.

Para tener acceso a este asistente, abra algún archivo de solución de Visual J++ 6.0 (.sln) o un archivo de proyecto (.vjp) en Visual J#. Si la solución de Visual J++ tiene más de un proyecto, el asistente se inicia para cada proyecto.

A continuación, el asistente le guía por una serie de pasos para identificar problemas en la actualización y ejecución de la aplicación en Visual J#. Los problemas detectados se muestran en un informe de actualización. Para obtener más información, vea [Informe de actualización de Visual J#](#).

El asistente crea un proyecto nuevo en el mismo lugar que el proyecto original y agrega los archivos de recursos del proyecto de Visual J++ al proyecto nuevo. No se realizan cambios en el código fuente. El asistente analiza los archivos de código fuente para determinar si existen problemas para actualizar el proyecto a Visual J#. Los resultados del análisis se utilizan para definir las propiedades de proyecto correspondientes en el proyecto actualizado. Todos los problemas de compatibilidad y de funcionalidad incompatible detectados en el análisis se agregan al informe de actualización.

El asistente intenta agregar referencias a los componentes COM y .NET que ha detectado que utilizan el proyecto. También actualiza los archivos de recursos (.resources y .properties) del proyecto al formato de archivo de recursos de .NET Framework (.resx).

En la primera página del asistente, puede elegir entre las siguientes opciones:

## Cancelar

Cancela la actualización a Visual J#.

## Siguiente

Pasa a la segunda página del asistente.

## Finalizar

Actualiza el proyecto inmediatamente, sin analizar los archivos de proyecto. Deberá realizar manualmente los cambios necesarios en las propiedades y en los archivos de código fuente del proyecto nuevo, para poder generar y ejecutar el proyecto actualizado.

Para obtener ayuda sobre el uso del asistente, presione F1 mientras se muestra el asistente.

## En esta sección

### [Asistente para actualización: elegir un tipo de proyecto](#)

Proporciona información detallada sobre el uso del asistente para elegir un tipo de proyecto.

### [Asistente para actualización: agregar referencias de proyecto](#)

Proporciona información detallada sobre el uso del asistente para agregar referencias de proyecto.

## Secciones relacionadas

### [Actualizar proyectos de Visual J++ 6.0](#)

Explica cómo actualizar proyectos de Visual J++ 6.0 a Visual J# utilizando el Asistente para actualización.

### [Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#)

Trata el procedimiento para convertir los recursos que utiliza un proyecto de Visual J++ 6.0 en recursos de .NET Framework, incluidos los problemas de compatibilidad que supone la incrustación o vinculación de recursos.

### [Informe de actualización de Visual J#](#)

Describe el formato y el propósito del informe de actualización.

### [Referencia para la actualización a Visual J#](#)

Proporciona ayuda para problemas detectados por el Asistente para actualización durante la actualización de proyectos de Visual J++ 6.0 a Visual J#.

# Asistente para actualización: elegir un tipo de proyecto

El Asistente para actualización a Visual J# intenta determinar el tipo de proyecto al que se debe actualizar el proyecto de Visual J++ . En los casos en que el proyecto se puede actualizar a más de un tipo, puede elegir el tipo de proyecto que desea crear.

En esta página del asistente, seleccione el tipo de proyecto y haga clic en **Siguiente** para continuar el asistente. Si desea actualizar el proyecto inmediatamente y omitir el resto de pasos del asistente, haga clic en **Finalizar**.

Las opciones de tipo de proyecto son:

## Aplicación de consola

Si el proyecto de Visual J++ estaba pensado para ejecutarlo como una aplicación de línea de comandos utilizando la herramienta Jview.exe, se debería actualizar a una aplicación de consola en Visual J#. Si piensa utilizar las clases de este proyecto en otro proyecto, debería considerar la creación del nuevo proyecto como una biblioteca de clases. Por ejemplo, debe elegir **Biblioteca de clases** si va a actualizar un proyecto que contenga una biblioteca de clases o un control. Sólo los proyectos con bibliotecas de clases pueden tener referencias a su resultado en otros proyectos en Visual Studio .NET.

## Aplicación para Windows

Si el proyecto de Visual J++ se creó como una aplicación que utiliza clases WFC o AWT, se debería actualizar a una aplicación para Windows en Visual J#. Si piensa utilizar las clases de este proyecto en otro proyecto, debería considerar la creación del nuevo proyecto como una biblioteca de clases.

## Biblioteca de clases

Si el proyecto de Visual J++ se creó de forma que sus clases se pudieran utilizar en otros proyectos, o si se creó como una DLL COM, se debería actualizar a una biblioteca de clases en Visual J#. Un proyecto de control de Visual J++ se debe actualizar también a un proyecto de biblioteca de clases.

Si desea cambiar el tipo de proyecto después de la actualización, puede cambiar las propiedades del proyecto en Visual J# para seleccionar el tipo de proyecto manualmente. Para obtener más información, vea

[Establecer las propiedades de un proyecto de Visual J#](#).

## Vea también

[Asistente para actualización a Visual J#](#)

# Asistente para actualización: agregar referencias de proyecto

El Asistente para actualización a Visual J# intenta analizar los archivos de proyecto de Visual J++ 6.0 para determinar las propiedades del proyecto de Visual J# actualizado, además de identificar las API no compatibles y otros problemas de compatibilidad. Para analizar los recursos totalmente, el asistente necesita referencias a todas las clases utilizadas en el proyecto de Visual J++ 6.0 que no están definidas en el proyecto en sí ni en bibliotecas de clases de Visual J#. Vea [Compatibilidad con bibliotecas de clases](#) para obtener una lista de las clases compatibles con Visual J#. Para que el análisis sea completo, se necesitan también referencias a las bibliotecas de tipos COM que describen clases o interfaces COM utilizadas en el proyecto.

Si el proyecto no tiene referencias externas y elige no agregarlas utilizando esta página del asistente, el asistente puede actualizar el proyecto de Visual J++ 6.0 a Visual J#. Sin embargo, para generar y ejecutar el proyecto actualizado, puede ser necesario agregar referencias externas al proyecto más adelante.

El valor de CLASSPATH definido para el proyecto de Visual J++, si lo hay, se muestra en esta página del asistente. A continuación, puede identificar las ubicaciones de las referencias externas.

Haga clic en el botón **Agregar** para seleccionar uno o más componentes .NET (también denominados ensamblados o bibliotecas de clases .NET) o bibliotecas de tipos COM para agregarlos a la lista de referencias. Utilice el botón **Quitar** para quitar elementos de la lista.

Los componentes .NET y COM agregados a la lista de referencias se agregarán también a la lista de referencias del proyecto actualizado.

Se recomienda agregar todas las referencias conocidas a la lista antes de la actualización. Si no agrega elementos a la lista de referencias, el análisis de actualización intentará identificar las referencias externas no resueltas en el proyecto, como se muestra a continuación:

- Se buscarán referencias COM en el Registro.
- Se buscarán referencias a clases en la variable CLASSPATH definida en el proyecto de Visual J++.

Si el proyecto de Visual J++ utiliza clases de otra biblioteca o proyecto, debe agregarles una referencia. En Visual J++ 6.0, el valor de CLASSPATH del proyecto se utiliza para buscar clases a las que se hace referencia. Si alguna de estas clases no forma parte de las bibliotecas de clases de Visual J#, deberá agregar una referencia a una biblioteca de clases (.dll) que contenga implementaciones de las clases a las que se hace referencia.

Si tiene los archivos de código fuente para las clases a las que se hace referencia, puede generarlos para crear una DLL utilizando Visual J#.

Si no tiene los archivos de código fuente, puede convertir los archivos de clase (.class) o de almacenamiento (.cab, .jar, o .zip) en bibliotecas de clases utilizando el [Conversor binario de Visual J#](#).

Si el proyecto de Visual J++ utiliza clases o interfaces definidas en una DLL COM o una biblioteca de tipos, puede agregar el archivo DLL o de biblioteca de tipos (.tlb) a la lista de referencias.

Si las clases del proyecto extienden o implementan clases e interfaces definidas externamente y las referencias a ellas no están disponibles, es posible que el análisis sea incompleto para esas clases.

## Vea también

[Asistente para actualización a Visual J#](#) | [Compatibilidad con bibliotecas de clases](#) | [Conversor binario de Visual J#](#)

# Informe de actualización de Visual J#

El [Asistente para actualización a Visual J#](#) crea un informe de actualización por cada proyecto que se actualiza de Visual J++ 6.0 a Visual J#. Este informe se agrega al proyecto actualizado. De manera predeterminada, el informe de actualización se abre cuando se cierra el asistente. Este informe contiene información relativa al proceso de actualización, así como una lista de problemas que deben solucionarse para poder ejecutar el proyecto.

El informe de actualización tiene formato HTML; para verlo dentro del entorno de desarrollo de Visual Studio .NET, haga doble clic en el archivo en el Explorador de soluciones o utilice un explorador externo eligiendo **Explorar con** en el menú **Archivo**.

El informe incluye tres secciones:

- La primera opción contiene información general sobre la actualización, incluida la ubicación del proyecto actualizado y la hora de la actualización.
- La segunda parte contiene una lista de problemas identificados durante la actualización. Si extiende la sección de cada categoría, verá los detalles de los problemas, incluido el impacto del problema y el número de veces que aparece en el proyecto. Puede expandir más aún cada problema para ver información importante, como los archivos donde se ha detectado. Cada categoría y descripción de problema es un vínculo a un tema de ayuda con más información para solucionar el problema.
- La última parte contiene información de actualización adicional, como una lista de referencias agregadas al proyecto actualizado.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#) | [Asistente para actualización a Visual J#](#) | [Referencia para la actualización a Visual J#](#)

# Actualizar proyectos de Visual J++ 6.0

Las soluciones y los proyectos de Visual J++ 6.0 se pueden actualizar fácilmente a Visual J# utilizando el [Asistente para actualización a Visual J#](#). Si se abre una solución de Visual J++ en Visual Studio .NET, el sistema le pedirá que actualice cada proyecto utilizando el Asistente para actualización. Se generará un informe de actualización que se agregará a cada proyecto actualizado. Este informe enumera los problemas de actualización detectados durante el análisis de los archivos de proyecto por el Asistente para actualización.

**Nota** Ningún archivo de código fuente del proyecto se modifica durante la actualización.

Antes de comenzar la actualización, es preciso cerrar la solución si está abierta en Visual J++ 6.0. Asimismo, si el proyecto está bajo control de código fuente, el archivo de solución debe tener acceso de escritura.

## Para actualizar un proyecto de Visual J++ 6.0 a Visual J#

1. Abra el proyecto en Visual J++ 6.0. Defina la configuración activa como **Lanzamiento**. Genere el proyecto. Ejecute la aplicación y asegúrese de que no haya errores.
2. Cierre Visual J++ 6.0 e inicie Visual Studio .NET (con Visual J# instalado). En el menú **Archivo**, seleccione **Abrir** y haga clic en **Proyecto**.
3. En el cuadro de diálogo **Abrir proyecto**, busque la ubicación del proyecto de Visual J++ y seleccione el archivo de proyecto (.vjp) o de solución (.sln). Esto iniciará el Asistente para actualización.
4. En la primera página del asistente, haga clic en **Finalizar** si desea omitir el análisis del proyecto. Se recomienda continuar con todos los pasos del asistente.

**Sugerencia** Para obtener más información, presione **F1** mientras está en el asistente.

5. En la segunda página del asistente, elija entre los tipos de proyecto disponibles para actualizar el proyecto.

**Sugerencia** Para obtener una explicación de las opciones disponibles, haga clic en el vínculo **Más información** de la esquina inferior derecha de la página.

6. En la tercera página del asistente, agregue referencias a bibliotecas COM y componentes .NET que contengan clases utilizadas en el proyecto.

**Sugerencia** Para obtener más información sobre la adición de referencias, haga clic en el vínculo **Más información** de la esquina inferior derecha de la página.

7. Haga clic en **Siguiente** para comenzar el análisis de los archivos de proyecto con el fin de detectar problemas de actualización. Esto puede tardar varios minutos, dependiendo del tamaño del proyecto.
8. Cuando termine el análisis, el asistente mostrará la página siguiente. Haga clic en **Finalizar** para completar el proceso de actualización. El proyecto actualizado aparecerá en el Explorador de soluciones.
9. El informe de actualización se abre de manera predeterminada. Si no se abre, haga doble clic en el nodo `_UpgradeReport.htm` en el Explorador de soluciones para verlo.
10. Revise el informe de actualización. Corrija todos los problemas que darían lugar a errores durante la generación y en tiempo de ejecución.

**Sugerencia** Haga clic en la descripción de cada problema para ver un tema de ayuda relacionado.

11. Presione F5 para generar y ejecutar el proyecto actualizado.

Algunos de los valores del proceso de generación que se configuran de manera independiente para las versiones de depuración y lanzamiento en Visual J++ 6.0 son ahora propiedades comunes en Visual J#. Por ejemplo:

- Tipo de resultado indica una aplicación para Windows, una aplicación de consola o una biblioteca de clases.
- Objeto inicial indica la clase a cuyo método se llama cuando se inicia la aplicación.

Cuando se actualizan proyectos a Visual J#, los valores de las propiedades anteriores se toman de la configuración Lanzamiento del proyecto de Visual J++. Para obtener ayuda con algunos problemas comunes de los proyectos actualizados, vea [Problemas comunes al actualizar proyectos de Visual J++ 6.0](#).

Durante el proceso de actualización se agrega la extensión .old al archivo de solución de Visual J++, para disponer de una copia de seguridad. Por ejemplo, vj6App.sln tendrá ahora como nombre vj6App.sln.old. El nuevo archivo de solución creado por el proceso de actualización tiene el mismo nombre que el archivo de solución de Visual J++ inicial, es decir, vj6App.sln; los nuevos



archivos de proyecto tendrán la extensión .vjsproj.

#### **Para usar la solución original (copia de seguridad) de Visual J++ en Visual J++**

1. Cambie el nombre al nuevo archivo de solución actualizado, por ejemplo, vj6App.new.sln.
2. Cambie el nombre a cualquier archivo con extensión .suo. Por ejemplo, vj6App.suo se denominará vj6App.new.suo.

**Nota** En Windows los archivos con extensión .suo están ocultos de forma predeterminada.

3. Quite .old de la extensión .sln.old anexada al archivo de solución original. Por ejemplo, vj6App.sln.old se denominará vj6App.sln.
4. Abra en Visual J++ el archivo de solución obtenido en el paso 3.

**Nota** El asistente puede no ser capaz de detectar todos los aspectos en la actualización, especialmente si no se especificaron las referencias requeridas en la página 3 del asistente.

#### **Vea también**

[Asistente para actualización a Visual J#](#) | [Informe de actualización de Visual J#](#) |

[Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#) | [Actualizar proyectos de Visual J++ 6.0](#)

# Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos

Para poder incrustar o vincular un recurso de Visual J++ existente en ensamblados administrados para utilizarlos en Common Language Runtime, se deben convertir los archivos de recursos al formato compatible con .NET Framework.

Si bien la extensión de archivo para archivos de recursos en .NET Framework y Visual J++ 6.0 es .resources, los dos formatos son en realidad muy diferentes y no son intercambiables.

## Convertir recursos

Cuando se actualiza un proyecto de Visual J++, el [Asistente para actualización a Visual J#](#) convierte automáticamente los archivos de recursos de Visual J++ (.resources y .properties) asociados al proyecto actualizado en archivos de recursos de .NET Framework (.resx). Cuando se genera el proyecto más adelante en Visual J#, los archivos .resx se convierten en archivos de recursos de .NET Framework (.resources) y se incrustan en el resultado del proyecto.

El Asistente para actualización no manipula automáticamente los archivos de recursos que tienen formatos diferentes a .resources o .properties (como .bmp o .gif). Estos archivos de recursos se deben convertir al formato .resx y agregar al proyecto actualizado manualmente.

Los pasos siguientes muestran cómo convertir manualmente archivos de recursos de proyectos de Visual J++ 6.0 en archivos de recursos de .NET Framework.

### Para convertir archivos de recursos manualmente

1. Identifique todos los archivos de recursos utilizados en el proyecto de Visual J++.
2. Convierta estos archivos de recursos al formato de .NET Framework (.resx) utilizando la herramienta [Vjsresgen.exe](#) que se proporciona con los ejemplos de Visual J#.
3. Agregue los archivos .resx generados al proyecto actualizado.
4. Para compilar un proyecto desde la línea de comandos, convierta los archivos de recursos con formato .resx al formato .resources utilizando la herramienta [Generador de archivos de recursos \(Resgen.exe\)](#) de .NET Framework. Esto prepara los archivos de recursos para incrustarlos o vincularlos a un ensamblado administrado.
5. Especifique la opción `/resource` del compilador de Visual J# para incrustar el archivo de recursos en el ensamblado administrado. Si la aplicación sólo está disponible en formato .class, use [Conversor binario de Visual J#](#) y especifique la opción `/linkresource`.

## Problemas de compatibilidad de recursos con Visual J++ 6.0

En Visual J++ 6.0, cada archivo de recursos está asociado a una clase. Los archivos de recursos asociados a una clase se deben incrustar o vincular en el mismo ensamblado que contiene la clase.

- En Visual J#, si se modifica el archivo de recursos de Visual J++ original, es necesario crear de nuevo el archivo de recursos de .NET Framework y agregarlo al ensamblado.
- No se admite la generación y el consumo dinámicos de recursos.
- Los archivos de recursos asociados utilizando la opción `/win32res` del compilador no son accesibles con la clase **Class.getResource** ni el método **Class.getResourceAsStream**.
- Los archivos de recursos asociados a una clase se deben incrustar o vincular en el mismo ensamblado que contiene la clase. Si varias clases comparten un único archivo de recursos, dicho archivo debe estar incrustado o vinculado en cada uno de los ensamblados en los que están incorporadas las clases.

Para obtener más información sobre las opciones del compilador de Visual J# utilizadas para incrustar o vincular recursos en ensamblados administrados, vea [Opciones del compilador de Visual J#](#). Para obtener más información sobre las opciones del Conversor binario de Visual J# (Jblmp.exe), vea [Conversor binario de Visual J#](#).

Para obtener más información sobre Resgen.exe, vea [Generador de archivos de recursos \(Resgen.exe\)](#).

## Vea también

[Utilizar recursos en aplicaciones de Visual J#](#) | [Asistente para actualización a Visual J#](#) | [Actualizar proyectos de Visual J++ 6.0](#) | [Actualizar proyectos de Visual J++ 6.0](#)

# Actualizar proyectos mediante la interoperabilidad de Java y COM

En esta sección se explica cómo compilar, depurar y ejecutar componentes de Java o COM desarrollados con Visual J++® 6.0 en Visual J#.

## En esta sección

### [Información general: actualizar componentes de Java o COM](#)

Proporciona información general acerca de los proyectos de Visual J++ 6.0 que utilizan la interoperabilidad de Java y COM, y cómo actualizarlos.

### [Antecedentes: interoperabilidad COM en Visual J++ 6.0](#)

Explica el acceso a componentes COM desde Java y viceversa.

### [Antecedentes: interoperabilidad COM en Common Language Runtime](#)

Explica el acceso a componentes COM desde código administrado y el acceso a componentes administrados desde COM.

### [Actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM](#)

Proporciona todos los pasos e información detallada sobre cómo compilar y ejecutar en Visual J# una aplicación de Visual J++ 6.0 que tiene acceso a componentes COM.

### [Actualizar componentes de Visual J++ 6.0 expuestos a COM](#)

Proporciona todos los pasos e información detallada sobre cómo compilar y ejecutar en Visual J# una aplicación de Visual J++ 6.0 a la que tienen acceso componentes COM.

### [Escenarios no compatibles con Visual J#](#)

Trata los escenarios de Java y COM que no son compatibles con Visual J#.

### [Problemas conocidos y soluciones](#)

Ofrece soluciones para escenarios de Java y COM.

## Secciones relacionadas

### [Interoperar con código no administrado](#)

Describe los servicios de interoperabilidad proporcionados por Common Language Runtime.

### [Actualizar proyectos de Visual J++ 6.0](#)

Proporciona recomendaciones y procedimientos para actualizar proyectos de Visual J++ 6.0 a Visual J#.

### [Ejemplo de COMCallsJava](#)

Muestra la actualización de componentes de Java de Visual J++ 6.0 expuestos como archivos DLL COM.

### [Ejemplo de JavaCallsCOM](#)

Muestra la actualización de aplicaciones de Java o COM de Visual J++ 6.0 a Visual J#.

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

# Información general: actualizar componentes de Java o COM

Visual J# admite la interoperabilidad entre aplicaciones administradas y componentes COM. Esta compatibilidad incluye la capacidad de compilar y ejecutar componentes de Visual J++ 6.0 existentes, así como código más nuevo que utilice semántica de interoperabilidad de .NET Framework.

Por tanto, las aplicaciones administradas y los componentes COM pueden tener acceso a sus servicios mutuamente. Como consecuencia, los desarrolladores pueden combinar la sintaxis de Java con las ventajas de COM y desarrollar aplicaciones que interactúen con componentes, aplicaciones y servicios del sistema operativo escritos en cualquier lenguaje compatible con COM en cualquier plataforma.

Para las aplicaciones nuevas, el modo recomendado para interoperar entre los componentes COM y el código administrado consiste en utilizar la semántica de interoperabilidad proporcionada por Common Language Runtime y las herramientas. Para obtener más información acerca del uso de la semántica de Common Language Runtime para la interoperabilidad COM, vea [Interoperar con código no administrado](#).

Visual J# admite la compilación y ejecución de componentes de Java y COM de Visual J++ 6.0. Esto incluye:

- Compatibilidad con la mayoría de los escenarios y características de Java y COM disponibles en Visual J++ 6.0 y Microsoft® SDK para Java.
- Compatibilidad de código fuente para aplicaciones de Java y COM de Visual J++ 6.0, de forma que se puedan compilar y ejecutar con muy pocos cambios o ningún cambio en el código fuente. Esto incluye la capacidad de compilar y ejecutar contenedores generados por la herramienta JActiveX™ en aplicaciones de Visual J++ 6.0 existentes y las directivas @com encontradas en estos contenedores.

**Nota** Visual J# no garantiza compatibilidad total de código fuente para compilar y ejecutar todas las aplicaciones de Java y COM de Visual J++ 6.0 existentes. Vea [Escenarios no compatibles con Visual J#](#) para obtener más información.

## Vea también

[Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

# Antecedentes: interoperabilidad COM en Visual J++ 6.0

Visual J++ 6.0 admitía dos escenarios de uso principales en la interoperabilidad de Java y COM.

## En esta sección

### [Obtener acceso a componentes COM desde Java](#)

Explica cómo utilizar contenedores Java a los que se puede llamar (JCW, Java Callable Wrappers) para tener acceso a componentes COM desde el código de Java.

### [Obtener acceso a componentes de Java desde COM](#)

Explica la compatibilidad de Visual J++ 6.0 con la compilación y el registro de clases de Java como componentes COM.

## Secciones relacionadas

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Trata las consideraciones que se deben tener en cuenta a la hora de actualizar proyectos de Visual J++ 6.0 que utilicen la interoperabilidad de Java y COM.

# Obtener acceso a componentes COM desde Java

Para tener acceso a un componente COM, ejecute la herramienta JActiveX en una biblioteca de tipos COM (archivo .tlb) o una DLL COM que contenga una biblioteca de tipos incrustada. La herramienta JActiveX, que se distribuye con Visual J++ 6.0 y Microsoft SDK para Java, genera contenedores para todas las coclases e interfaces COM de la biblioteca de tipos. Estos contenedores, denominados contenedores de Java a los que se puede llamar (JCW, Java Callable Wrappers), actúan como un proxy para los objetos COM reales y contienen directivas **@com** que el compilador de Java (jvc.exe) comprende. A continuación, se importan estos contenedores en una aplicación y se compilan junto con el código de la aplicación. Las directivas **@com** de los contenedores se compilan en atributos en los archivos .class y Microsoft Java Virtual Machine (MSJVM) los interpreta en tiempo de ejecución.

Para crear una instancia de un objeto COM, llame a **new** en el contenedor adecuado. El objeto se puede convertir entonces en las interfaces y se puede llamar a los métodos que contienen. MSJVM utiliza las directivas @com de los contenedores generados para hacer lo siguiente:

- Ubicar y llamar a **CoCreateInstance** en la coclase COM.
- Traducir las llamadas a métodos del contenedor como llamadas del objeto COM real.
- Administrar las referencias al objeto COM.
- Calcular las referencias de los parámetros entre COM y Java.

Los desarrolladores de Java utilizarán el objeto COM como si estuvieran utilizando cualquier otro objeto de Java y sin preocuparse por ningún detalle, igual que haría un desarrollador de COM típico.

También puede alojar controles ActiveX en Java.

## Vea también

[Antecedentes: interoperabilidad COM en Visual J++ 6.0](#)

# Obtener acceso a componentes de Java desde COM

Visual J++ 6.0 admite proyectos que compilen clases de Java en DLL COM o controles ActiveX (OCX). Estos DLL COM u OCXs ActiveX se registran y utilizan en herramientas como Visual Basic 6.0 y Visual C++ 6.0. De manera alternativa, se pueden utilizar herramientas de la línea de comandos, como VJReg y JavaReg, que se distribuye con Visual J++ 6.0 y Microsoft SDK para Java, con el fin de registrar clases de Java que contengan el atributo **@com.register** como controles ActiveX o COM, de manera que Visual Basic 6.0 y Visual C++ 6.0 puedan tener acceso a las mismas.

Para obtener más información sobre la compilación y ejecución de componentes de Java y COM, vea los textos relacionados en la documentación de Visual J++ 6.0 (MSDN) o Microsoft SDK para Java. Para obtener más información acerca del acceso a la documentación de Visual J++ 6.0, vea [Buscar información de referencia sobre Java y JDK](#).

## Vea también

Antecedentes: interoperabilidad COM en Visual J++ 6.0 | [http://www.microsoft.com/java/resource/java\\_com2.htm](http://www.microsoft.com/java/resource/java_com2.htm) | [http://www.microsoft.com/java/resource/java\\_com.htm](http://www.microsoft.com/java/resource/java_com.htm)

# Antecedentes: interoperabilidad COM en Common Language Runtime

Se puede tener acceso a componentes COM desde clientes administrados; y también se puede tener acceso a un servidor administrado desde clientes COM mediante la semántica de interoperabilidad COM de .NET Framework.

## En esta sección

### [Obtener acceso a componentes COM desde código administrado](#)

Explica cómo utilizar los contenedores a los que se puede llamar en tiempo de ejecución (RCW) para tener acceso a componentes COM desde código administrado.

### [Obtener acceso a componentes administrados desde COM](#)

Explica cómo utilizar la herramienta Regasm.exe para registrar una DLL administrada como un componente COM.

## Secciones relacionadas

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Trata las consideraciones que se deben tener en cuenta a la hora de actualizar proyectos de Visual J++ 6.0 que utilicen la interoperabilidad de Java y COM.



# Obtener acceso a componentes COM desde código administrado

Al ejecutar la herramienta Importador de la biblioteca de tipos (Tlbimp.exe) en una biblioteca de tipos COM o en una DLL COM con una biblioteca de tipos incrustada, se generan contenedores administrados para el componente COM. La herramienta Tlbimp.exe se distribuye con .NET Framework SDK y Visual Studio .NET. A continuación, se puede hacer referencia a estos contenedores en cualquier aplicación administrada que requiera acceso a las clases de la DLL COM.

Los contenedores generados se denominan contenedores a los que se puede llamar en tiempo de ejecución (RCW, Runtime Callable Wrappers) y actúan como un proxy para los objetos COM reales. Estos RCW contienen determinados atributos de interoperabilidad de .NET Framework que Common Language Runtime interpreta para crear instancias de los objetos COM reales en tiempo de ejecución.

## Obtener acceso a componentes COM desde código administrado

1. Para la DLL COM a la que se desee tener acceso desde una aplicación de Visual J#, hay que generar contenedores administrados desde la biblioteca de tipos COM utilizando Tlbimp.exe como se muestra a continuación:

```
tlbimp /out:FB.dll My.tlb
```

2. Se crean referencias a los contenedores generados en la aplicación. Ahora se puede tener acceso a los componentes COM llamando a **new** en el contenedor adecuado. Al compilar los archivos de código fuente, hay que crear referencias a los contenedores generados como se indica a continuación:

```
vjc /reference:FB.dll application_sources
```

3. Para crear una instancia de un objeto COM, hay que utilizar **new** en el contenedor administrado correspondiente a la coclase. Después se puede convertir el objeto en una interfaz adecuada y llamar a métodos en la misma. Common Language Runtime utiliza los atributos de interoperabilidad de los contenedores para buscar y llamar a **CoCreateInstance** en la coclase COM, traducir llamadas a métodos del contenedor en llamadas del objeto COM real, administrar referencias al objeto COM y convertir los parámetros de los tipos COM en tipos de .NET Framework y viceversa.

Esta compatibilidad también incluye la capacidad de alojar controles ActiveX en formularios Windows Forms.

## Vea también

[Antecedentes: interoperabilidad COM en Common Language Runtime](#)

# Obtener acceso a componentes administrados desde COM

Se puede utilizar la herramienta Registro de ensamblados (Regasm.exe) para registrar una DLL administrada como un componente COM y, por tanto, permitir a herramientas como Visual Basic y Visual C++ tener acceso al componente administrado como una DLL COM o un control ActiveX. Además, se puede utilizar la herramienta Exportador de la biblioteca de tipos (Tlbexp.exe) o la opción /tlb de Regasm para generar una biblioteca de tipos para el componente administrado. Estas herramientas se distribuyen con .NET Framework SDK y Visual Studio .NET.

## Pasos básicos para tener acceso a componentes administrados desde COM

1. Para registrar el componente como un servidor COM, hay que utilizar la herramienta Regasm.exe de .NET Framework:

```
Regasm /tlb:mytlb.tlb /codebase managedobj.dll
```

2. La clase administrada que se va a registrar como componente COM es ahora accesible desde clientes no administrados.

Para obtener más información sobre la interoperabilidad COM utilizando la semántica de .NET Framework, vea [Interoperar con código no administrado](#).

## Vea también

[Antecedentes: interoperabilidad COM en Common Language Runtime](#)

# Actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM

En Visual J#, se puede compilar y ejecutar una aplicación de Java o COM de Visual J++ 6.0 que tenga acceso a componentes COM. En los siguientes pasos se muestra cómo actualizar manualmente un componente de Java o COM desde el símbolo del sistema. Al actualizar proyectos de Visual J++ en Visual Studio .NET, el [Asistente para actualización a Visual J#](#) automatiza muchos de los pasos.

## Para actualizar una aplicación a Visual J#

1. Hay que generar contenedores administrados para la biblioteca de tipos del componente COM utilizando la herramienta Tlbimp.exe. Normalmente, esto se hace del siguiente modo:

```
tlbimp.exe /out:managed_WrapperDLL COMComponent.tlb
```

2. Se compila la aplicación de Java junto con los contenedores JActiveX, haciendo referencia al contenedor administrado generado por Tlbimp. Para ello, hay que utilizar el compilador de Visual J#, vjc.exe. Normalmente, esto se hace del siguiente modo:

```
vjc ApplicationSources JActiveXWrapperSources /r:managed_WrapperDLL
```

3. Para ejecutar la aplicación, hay que ejecutar el archivo .exe generado.
4. La aplicación se puede depurar utilizando las herramientas de depuración disponibles en .NET Framework SDK, CorDbg.exe o DbgClr.exe; o bien, utilizando el depurador de Visual Studio .NET. Para obtener más información sobre estas herramientas, vea [Interoperar con código no administrado](#) y [Depurar](#).

Los contenedores administrados generados por Tlbimp no son compatibles con los contenedores JActiveX. Por tanto, toda intención de descartar los contenedores JActiveX e importar directamente los contenedores administrados en la aplicación supondría importantes cambios en el código fuente de la aplicación. Por ejemplo, los métodos de los contenedores administrados generados por Tlbimp aceptan los tipos de .NET Framework como parámetros de método y valores devueltos, mientras que los contenedores generados por la herramienta JActiveX aceptan los tipos de Java como parámetros de método y valores devueltos. Entre estos contenedores hay otras diferencias como el modo en que se exponen las propiedades y los eventos, cómo se notifican los tipos HRESULT, y las interfaces que implementan los contenedores.

Al actualizar aplicaciones de Java y COM de Visual J++ 6.0 a Visual J#, modificando manualmente los contenedores generados por JActiveX para agregar API o interoperabilidad COM de .NET Framework, pueden surgir problemas inesperados en tiempo de ejecución. Mezclar estos atributos de Visual J++ 6.0, como **@com** y **@dll**, con atributos de interoperabilidad COM de .NET Framework no sólo no se admite, sino que se desaconseja encarecidamente tanto en aplicaciones Java y COM de Visual J++ 6.0 como en las aplicaciones nuevas escritas con Visual J#.

## En esta sección

### [Información general sobre el diseño: obtener acceso a componentes COM](#)

Trata sobre las características de diseño que permiten actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM para ejecutarlas en .NET Framework.

### [Controlar el modelo de subprocesamiento de los componentes COM](#)

Explica las diferencias entre Microsoft Java Virtual Machine (MSVJM) y .NET Framework Common Language Runtime en cuanto al control del modelo de subprocesamiento para objetos COM

## Secciones relacionadas

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Trata las consideraciones que se deben tener en cuenta a la hora de actualizar proyectos de Visual J++ 6.0 que utilicen la interoperabilidad de Java y COM.

# Información general sobre el diseño: obtener acceso a componentes COM

Puesto que los objetivos fundamentales eran mantener la compatibilidad de código fuente y utilizar la capacidad de interoperabilidad que proporciona Common Language Runtime, los componentes de Java y COM de Visual J++ 6.0 se compilan y ejecutan en Visual J# utilizando los contenedores JActiveX y los contenedores administrados generados por Tlbimp.

Para actualizar una aplicación de Visual J++ 6.0, compílela junto a los contenedores generados por JActiveX utilizando vjc.exe, haciendo referencia a los ensamblados de contenedor administrado generados por Tlbimp. El compilador de Visual J# (vjc.exe) es compatible con @com. En tiempo de compilación, vjc.exe genera una implementación de los contenedores JActiveX que utilizan los contenedores administrados generados por Tlbimp. El compilador emite código de manera que cada contenedor JActiveX se conecta al contenedor administrado correspondiente y aloja una referencia a éste. Siempre que se crea una instancia nueva del contenedor JActiveX, se crea también una instancia nueva del contenedor administrado correspondiente. Esto, a su vez, desencadenará la creación de la coclase COM que realiza la infraestructura de interoperabilidad de .NET Framework.

Los miembros de los contenedores JActiveX se asignan a los miembros de los contenedores administrados. Por tanto, la invocación de cada método del contenedor JActiveX da lugar a que se invoque el método del contenedor administrado. Common Language Runtime invocará después al método correspondiente de la coclase COM. Esta asignación se lleva a cabo en tiempo de compilación utilizando los atributos @com de los contenedores JActiveX.

Las implementaciones de contenedores JActiveX se utilizan también para ocultar sin problemas las diferencias entre los dos contenedores. La implementación del contenedor JActiveX convierte los tipos de Java en tipos de .NET Framework antes de llamar a los métodos de los contenedores administrados. A continuación, se convierten en los tipos COM correspondientes en tiempo de ejecución. También se hace al contrario para los valores de método devueltos. Por tanto, estas aplicaciones continúan utilizando los contenedores generados por JActiveX, pero ahora utilizan la capacidad de Common Language Runtime para crear instancias de los objetos COM y llamar a los métodos que contienen.

La ventaja de este proceso es que la aplicación puede seguir utilizando los contenedores generados por JActiveX en su código.

## Componente de Java o COM después de la actualización a Visual J#



Los objetos COM se siguen exponiendo como objetos que se heredan de las mismas clases e interfaces com.ms.\* que en Visual J++ 6.0. Para la aplicación, estos contenedores tienen el mismo diseño y las mismas características de tiempo de ejecución que en Visual J++ 6.0. La conversión a interfaces COM y la llamada a los métodos de las interfaces funcionan igual que en Visual J++ 6.0.

Los tipos de Java pasados a métodos COM se convierten en los tipos COM adecuados; y los valores devueltos de los métodos COM se convierten en los tipos de Java adecuados. La asignación entre los dos tipos es similar a la de Visual J++ 6.0 e incluye interfaces como **IUnknown** e **IDispatch** y los tipos de datos como VARIANT, SAFEARRAY, struct, enum, y clases e interfaces definidas por el usuario.

El contenedor JActiveX inicia una excepción **com.ms.com.ComFailException** para los HRESULT de error devueltos por métodos COM.

La semántica para recibir eventos COM y tener acceso a las propiedades se mantiene sin cambios.

Los objetos de Java pueden implementar interfaces COM. Cuando se pasa uno de estos objetos a COM, se puede invocar a **QueryInterface** en el objeto para esa interfaz con el fin de que se devuelva el puntero correcto.

## Vea también

[Actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM](#)

# Controlar el modelo de subprocesamiento de los componentes COM

Al ejecutar componentes de Java o COM de Visual J++ 6.0 en Visual J#, hay que tener en cuenta algunas de las diferencias entre el modo en que Microsoft Java Virtual Machine (MSJVM) y .NET Framework Common Language Runtime controlan el modelo de subprocesamiento para los objetos COM. Puesto que la capacidad de interoperabilidad COM de Visual J# se sitúa en capas sobre la interoperabilidad COM de .NET Framework, estos problemas afectan a las aplicaciones Java y COM de Visual J#.

- [Calcular las referencias de interfaces COM a través de límites de apartamento](#)
- [Controlar los mensajes enviados a objetos STA](#)

## Vea también

[Actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM](#)

# Calcular las referencias de interfaces COM a través de límites de apartamento

A las interfaces COM de las que no se pueden calcular las referencias a través de límites de apartamento en un escenario "COM llama a COM" puro tampoco se les pueden calcular las referencias en Common Language Runtime a través de subprocesos con modelos de apartamento incompatibles. Es decir, si no se pueden calcular las referencias de una interfaz COM a través de límites de apartamento, no se puede llamar a los métodos del contenedor JActiveX correspondiente a esa interfaz desde un subproceso de un modelo de apartamento incompatible.

**Nota** Este escenario es compatible con Microsoft Java Virtual Machine (MSJVM) pero no con Common Language Runtime. MSJVM permite llamar a interfaces COM desde subprocesos de modelos de apartamento incompatibles y controla el cálculo de referencias de los punteros de interfaz, transparentes para el desarrollador de Java.

De manera predeterminada, todos los subprocesos de Common Language Runtime (y, por tanto, para aplicaciones de Visual J++ 6.0 actualizadas a Visual J#) pertenecen al apartamento multiproceso (MTA). En otras palabras, los subprocesos administrados se inicializan de manera predeterminada para admitir únicamente objetos multiproceso, dejando las cuestiones de sincronización y cálculo de referencias al desarrollador.

Por tanto, este problema adquiere importancia cuando se actualizan aplicaciones de Visual J++ 6.0 que aloja objetos COM STA (apartamento de un único subproceso). En estas aplicaciones, un intento de llamada a los métodos de cualquier interfaz **IUnknown** (vtable) pura implementada por el objeto COM iniciaría una excepción. Las referencias de las interfaces **IUnknown** puras no se pueden calcular a través de apartamentos. Puesto que todos los subprocesos pertenecen al MTA de manera predeterminada, una llamada a estas interfaces daría lugar a la excepción **com.ms.com.ComFailException**. La solución a este problema consiste en hacer posible el cálculo de referencias de la interfaz COM a través de apartamentos, quizá de alguno de los siguientes modos:

- Generar y registrar una DLL stub o un proxy desde el archivo IDL del componente COM. Esto se puede hacer con la herramienta MIDL.
- Hacer las interfaces duales o de envío y registrar la biblioteca de tipos para que COM permita el cálculo de referencias basado en bibliotecas de tipos. Esto sólo es posible cuando los tipos utilizados en las interfaces admiten automatización. Para obtener más información sobre el cálculo de referencias de interfaces COM a través de límites de apartamento, vea la documentación de MSDN.
- Hacer que el modelo de apartamento del subproceso administrado que aloja el objeto COM sea compatible con el modelo de subprocesamiento de ese objeto. Puesto que todos los subprocesos son MTA de manera predeterminada, se puede alojar un objeto COM STA definiendo el modelo de apartamento del subproceso como STA. Esto se puede hacer de dos maneras:
  - Especificando el atributo `/** @attribute System.STAThread () */` en la función main del punto de entrada. De esta forma, se convierte el subproceso main de la aplicación en un subproceso STA. Por ejemplo:

```
/** @attribute System.STAThread () */
public static void main(String args[])
{
    ...
    new STACOMObject();
    ...
}
```

- Llame a la propiedad **ApartmentState** de la clase **Thread** de .NET Framework para convertirla en STA. Esto sólo se puede hacer antes de que se realicen llamadas a los componentes COM. Vea [/ApartmentState](#) para obtener más información.

```
System.Threading.Thread.GetCurrentThread().set_ApartmentState(System.Threading.ApartmentState.STA)
```

**Nota** Esta solución es específicamente necesaria cuando se actualizan aplicaciones WFC que alojan controles ActiveX de Visual J++ a Visual J#. En estos casos, hay que agregar el atributo `/** @attribute System.STAThreadAttribute () */` a la función main del punto de entrada si el subproceso main aloja el control ActiveX o utilizar la propiedad **ApartmentState** si cualquier otro subproceso aloja el control ActiveX.

**Vea también**

[Controlar el modelo de subprocesamiento de los componentes COM](#)

# Controlar los mensajes enviados a objetos STA

Si un subproceso STA que aloja un objeto COM STA se bloquea, Common Language Runtime suele llevar a cabo un bombeo implícito en nombre del objeto COM. Sin embargo, hay determinadas condiciones en las que Common Language Runtime no llevará a cabo este bombeo implícito:

- Si el subproceso se bloqueó por una llamada a **Thread.Sleep**.
- Si el subproceso cruzó el límite nativo administrado a través de invocación de plataforma o J/Direct y se bloqueó mientras estaba en código nativo.

En estos casos, no se entregarán mensajes al objeto COM STA.

La solución de este problema consiste en implementar un bombeo explícito de mensajes para ese subproceso utilizando las clases del paquete **com.ms.win32**, como se muestra en el ejemplo siguiente:

```
com.ms.win32.MSG msg = new com.ms.win32.MSG();
while (com.ms.win32.User32.GetMessage(msg, 0, 0, 0))
{
    com.ms.win32.User32.TranslateMessage(msg);
    com.ms.win32.User32.DispatchMessage(msg);
}
```

## Vea también

[Controlar el modelo de subprocesamiento de los componentes COM](#)



# Actualizar componentes de Visual J++ 6.0 expuestos a COM

Se puede compilar y ejecutar un componente de Java o COM de Visual J++ 6.0 al que se tenga acceso desde clientes COM. En los siguientes pasos se muestra cómo actualizar manualmente un componente de Java o COM desde el símbolo del sistema. Al actualizar proyectos de Visual J++ en Visual Studio .NET, el [Asistente para actualización a Visual J#](#) automatiza muchos de los pasos.

## Para actualizar una aplicación a Visual J#

1. Hay que compilar el componente de Java o COM con el compilador de Visual J# (vjc.exe). Por ejemplo:

```
vjc /target:library JavaSources
```

Hay que firmar el ensamblado para poder registrarlo en el siguiente paso utilizando la opción **/codebase** de [Regasm](#). Para firmar un ensamblado, hay que asociar el atributo [AssemblyKeyFile](#) que se encuentra en el espacio de nombres

**System.Reflection** a uno de los archivos del proyecto. Por ejemplo, `/** @assembly`

`System.Reflection.AssemblyKeyFile("myKey.snk") */`, donde `myKey.snk` es un archivo de pares de claves generado por la herramienta [sn.exe](#) de .NET Framework SDK

Si los clientes COM obtuvieran acceso al componente de Java o COM utilizando la biblioteca de tipos generada por la herramienta VJReg.exe, el cliente COM utilizaría la interfaz dispinterface expuesta en el componente y habría detectado los Displs de los miembros. No está garantizado que estos Displs de los miembros de clase sean los mismos en Visual J#. Esto puede dar lugar a que fallen los clientes COM existentes. Para evitarlo, si existe una biblioteca de tipos para el componente de Java o COM, hay que asegurarse de que está registrada en el equipo y de que se especifica el GUID de la biblioteca de tipos utilizando el parámetro typelib de la directiva **@com.register** (si no se encontró ya) para el componente. El compilador de Visual J# (vjc.exe) emitirá la advertencia VJS1553 en estos casos. Vea la documentación de VJS1553 para obtener más información.

2. Se registra el archivo DLL o EXE generado utilizando la herramienta Regasm.exe, que se distribuye con .NET Framework SDK y Visual Studio .NET. Por ejemplo:

```
Regasm /codebase generated_DLL_or_EXE
```

3. Si el componente de Java o COM se implementó utilizando las plantillas generadas por la herramienta JActiveX (utilizando la opción /c2j) o si el componente de Java implementa una interfaz de biblioteca de tipos, hay que hacer lo siguiente antes del primer paso:

- Utilizar la herramienta Tlbimp.exe (que se distribuye con .NET Framework SDK y Visual Studio .NET) para generar contenedores administrados desde la biblioteca de tipos que se utilizó para generar las plantillas JActiveX. Por ejemplo:

```
tlbimp.exe /keyfile:interopKey.snk COMComponent.tlb
```

Hay que tener en cuenta que el ensamblado administrado emitido se firma con un par de claves generado por la herramienta sn.exe de .NET Framework SDK. Esto evita la advertencia de que el ensamblado no tiene una firma segura cuando se ejecuta Regasm.

Se supone que existe una biblioteca de tipos para el componente de Java o COM. Cuando se crean DLL COM o proyectos de control, Visual J++ 6.0 crea automáticamente bibliotecas de tipos para el componente de Java expuesto a COM. También se puede crear una biblioteca de tipos para la clase de Java utilizando la herramienta VJReg.exe o JavaReg.exe, que se distribuye con Visual J++ 6.0 y Microsoft SDK para Java, si todavía no existe una biblioteca de tipos.

- A continuación, en el paso 1, hay que crear una referencia a los ensamblados de contenedor administrado generados en el paso anterior al compilar el componente de Java o COM.

```
vjc /r:TlbimpGeneratedWrappers JavaSources
```

Hay que firmar el ensamblado para poder registrarlo en el siguiente paso utilizando la opción **/codebase** de Regasm. Para firmar un ensamblado, hay que asociar el atributo **AssemblyKeyFileAttribute** que se encuentra en el espacio de nombres **System.Reflection** a uno de los archivos del proyecto. Por ejemplo, `/** @assembly`

`System.Reflection.AssemblyKeyFile("myKey.snk")` \*/, donde `myKey.snk` es un archivo de pares de claves generado por la herramienta `sn.exe` de .NET Framework SDK

Al actualizar aplicaciones de Java y COM de Visual J++ 6.0 a Visual J#, modificando manualmente los contenedores generados por JActiveX para agregar API o interoperabilidad COM de .NET Framework, pueden surgir problemas inesperados en tiempo de ejecución. Mezclar estos atributos de Visual J++ 6.0, como **@com** y **@dll**, con atributos de interoperabilidad COM de .NET Framework no sólo no se admite, sino que se desaconseja encarecidamente tanto en aplicaciones Java y COM de Visual J++ 6.0 como en las aplicaciones nuevas escritas con Visual J#.

## En esta sección

### [Información general sobre el diseño: acceso desde un componente COM](#)

Trata las características de diseño que permiten ejecutar actualizaciones de componentes de Java o COM en .NET Framework.

## Secciones relacionadas

### [Actualizar proyectos mediante la interoperabilidad de Java y COM](#)

Trata las consideraciones que se deben tener en cuenta a la hora de actualizar proyectos de Visual J++ 6.0 que utilicen la interoperabilidad de Java y COM.

# Información general sobre el diseño: acceso desde un componente COM

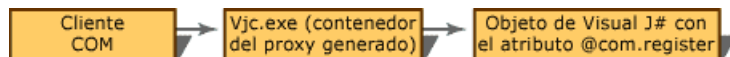
Para actualizar la aplicación de Visual J++ 6.0, hay que compilarla utilizando el compilador de Visual J# (vjc.exe). Durante la compilación, vjc.exe crea una clase contenedora equivalente que tiene los mismos campos, las mismas propiedades y los mismos métodos públicos que tenía una clase de Java expuesta a COM en Visual J++ 6.0. La diferencia consiste en que los métodos de la clase contenedora toman los tipos de .NET Framework equivalentes como parámetros y valores devueltos. El contenedor se marca como `ComVisible(true)`. Cuando se ejecuta la herramienta Regasm.exe en la DLL emitida, la clase contenedora se registra con el CLSID especificado en el atributo **@com.register** de la clase de Java, no en la clase de Java real en sí.

Los clientes COM interactúan directamente con el contenedor generado por vjc.exe en lugar de con el objeto de Java. Se puede tener acceso a los métodos utilizando secuencias de comandos o por medio de clientes COM que utilicen la interfaz **IDispatch** para invocar métodos. La estructura del contenedor y la biblioteca de tipos generada por esta clase son similares a lo que esperan los clientes COM de los objetos de Java en Visual J++ 6.0.

El contenedor crea internamente instancias del objeto de Java, asigna sus métodos a los del objeto de Java y controla todas las diferencias entre la clase de Java y la clase contenedora real expuesta, incluido el cálculo de referencias de los tipos.

La ventaja de este proceso es que la aplicación COM no tiene constancia de que el componente de Java se está ejecutando en Common Language Runtime en lugar de en Microsoft Java Virtual Machine (MSJVM). Para el cliente COM, el objeto de Java permanece sin cambios.

## Componente de Java o COM después de la actualización a Visual J#



Los tipos COM se convierten en los tipos de Java adecuados y los valores devueltos de métodos de Java se convierten en tipos COM. La asignación entre los dos tipos es similar a la de Visual J++ 6.0. Esto incluye interfaces como **IUnknown** e **IDispatch**, y tipos de datos como VARIANT, SAFEARRAY, struct, enum, y clases e interfaces definidas por el usuario.

Los HRESULT adecuados se devuelven al cliente COM cuando se inician excepciones de Java. La asignación entre las excepciones y los HRESULT debería ser la misma que en Visual J++.

## Vea también

[Actualizar componentes de Visual J++ 6.0 expuestos a COM](#)

# Escenarios no compatibles con Visual J#

En esta sección se tratan los escenarios no compatibles en las siguientes áreas:

- [Java llama a COM](#)
- [COM llama a Java](#)

## Vea también

[Actualizar proyectos mediante la interoperabilidad de Java y COM](#) | [Características de .NET Framework no compatibles](#) | [Bibliotecas de clases y API no compatibles](#) | [Problemas conocidos y soluciones](#)

# Java llama a COM

En esta sección se tratan los escenarios no compatibles en las siguientes áreas:

- [Compatibilidad con opciones de JActiveX](#)
- [Compatibilidad parcial con directivas @com](#)
- [Problemas del modelo de subprocesamiento](#)
- [Alojar controles ActiveX en Java](#)

## Vea también

[Escenarios no compatibles con Visual J#](#)

# Compatibilidad con opciones de JActiveX

Los contenedores creados con el Asistente para agregar contenedores COM de Visual J++ 6.0 se generan con las siguientes opciones de JActiveX:

```
/wfc /w /xi /X:rkc /l FileName.tmp /nologo /d ProjectName COMDLLNAME.dll
```

Estos contenedores se compilarán en el compilador de Visual J# (vjc.exe). En la siguiente tabla se muestran las opciones de JActiveX no compatibles.

Opción de JActiveX	Descripción	Resultado
<b>Traducción</b>		
/bx+	Expone controles ActiveX como Java Bean.	Error de compilación
/X:m-	Deshabilita el cálculo automático de referencias.	Se omite
/X:s2	Representa los enteros de 2 bytes sin signo como 'char'.	Se omite
/X:vi	Representa los punteros void como 'int'.	Se omite
<b>Salida</b>		
/ci:as	Crea entradas <b>EventInfo</b> para todas las interfaces del código fuente (sólo WFC).	Error de compilación
<b>Varios</b>		
/t:jnffile	Especifica un archivo JNF (archivo de información de biblioteca de tipos de Java) para la configuración personalizada (el valor predeterminado es ninguno).	Error de compilación
<b>Javatlb</b>		
/x2	Representa todos los enteros de 2 bytes como 'char'.	Se omite
/xh	No asigna S_FALSE a <b>ComSuccessException</b> .	Se omite
/n:jnffile	Especifica un archivo JNF para el cálculo personalizado de referencias (el valor predeterminado es ninguno).	Error de compilación
/G3.1	Microsoft Virtual Machine de destino 3.1 para Java (el valor predeterminado es /G4).	Se omite
/G4	Microsoft Virtual Machine de destino 4.0 para Java (el valor predeterminado es /G4).	

Los valores de la columna Resultado de la tabla se definen como sigue:

## Error de compilación

Puede que no se compilen los contenedores generados en vjc.exe.

## Se omite

Los contenedores generados se compilarán, pero se omitirá el código de estos contenedores específico para estas opciones. En tiempo de ejecución, esto puede dar lugar a problemas de cálculo de referencias y de incompatibilidad del modelo de subprocesamiento con respecto a Visual J++ 6.0.

Puede ocurrir que el compilador de Visual J# no compile los contenedores emitidos por versiones anteriores de las herramientas distribuidas antes de JActiveX.

La herramienta JActiveX no se distribuye con Visual J#. Se incluye con Visual J++ 6.0 y Microsoft SDK para Java. Para obtener más información sobre JActiveX, vea la documentación de Visual J++ 6.0 (MSDN) o de Microsoft SDK para Java. Para obtener más información acerca del acceso a la documentación de Visual J++ 6.0, vea [Buscar información de referencia sobre Java y JDK](#).

## Vea también

[Java llama a COM](#)

# Compatibilidad parcial con directivas @com

En la siguiente tabla se muestran los atributos de las directivas **@com** compatibles que se omiten.

Directivas @com	Atributos omitidos
@com.class	DynamicCasts
@com.interface	Thread
@com.method	addFlagsVtable
@com.parameters	customMarshalFlags, thread, type = CUSTOM   CUSTOMBYREF   CUSTOMBYVAL, custom Marshal, size, byref y array
@com.structmap	customMarshalFlags, customMarshal, addFlags, thread
@com.struct	safe, safeAddFlags
@com.register	typelib, version, description

Los contenedores que contienen algunas de las directivas o atributos anteriores se compilarán, pero pueden surgir problemas de incompatibilidad en tiempo de ejecución con respecto a Visual J++ 6.0, especialmente en el cálculo de referencias y en el subprocesamiento.

Como no se admiten los parámetros **customMarshalFlags**, **customMarshal** y **type = CUSTOM | CUSTOMBYREF | CUSTOMBYVAL** de las directivas **@com.parameters** y **@com.structmap**, Visual J# no admite el cálculo personalizado de referencias. El compilador de Visual J# no puede compilar contenedores generados por JactiveX que contengan directivas con estos parámetros.

Si se utilizan las macros **JTLBATTR\_IID\_IS** y **JTLBATTR\_SIZE\_IS** en el archivo IDL del componente COM para transmitir información de **iid\_is** y **size\_is** a la biblioteca de tipos, los contenedores generados por JactiveX contendrán directivas que utilizan los parámetros **type = CUSTOM** y **customMarshal**. Por tanto, el compilador de Visual J# no puede compilar estos contenedores. Para obtener más información sobre **JTLBATTR\_IID\_IS** y **JTLBATTR\_SIZE\_IS**, vea la documentación de Visual J++ 6.0 o Microsoft SDK para Java. Para obtener más información acerca del acceso a la documentación de Visual J++ 6.0, vea [Buscar información de referencia sobre Java y JDK](#).

## Vea también

[Java llama a COM](#)

# Problemas del modelo de subprocesamiento

Cuando se ejecutan componentes COM de Visual J++ 6.0 en Visual J#, hay algunos problemas conocidos de incompatibilidad del modelo de subprocesamiento. Para obtener más información, vea [Controlar el modelo de subprocesamiento de los componentes COM](#).

## Vea también

[Java llama a COM](#)



# Alojar controles ActiveX en Java

Sólo se admiten los contenedores JActiveX generados por controles ActiveX utilizando la opción **/wfc**. Ésta es la opción predeterminada que se utiliza cuando se arrastran controles ActiveX y se colocan en un formulario WFC del diseñador de formularios de Visual J++ 6.0.

## Vea también

[Java llama a COM](#)

# COM llama a Java

A continuación se enumeran las incompatibilidades:

- Existen incompatibilidades con Visual J++ 6.0 en el conjunto de interfaces expuestas por contenedores COM a los que se puede llamar (CCW) de componentes de Java. En Visual J#, los CCW no exponen las siguientes interfaces:
  - **IProvideClassInfo2** e **IExternalConnection**
  - **IPersist**, **IPersistStreamInit** e **IPersistStorage** (expuestas en MSJVM cuando una clase implementa **java.io.Serializable** o **java.io.Externalizable**)
- Las interfaces de marcador de la clase como **com.ms.com.NoAutoMarshaling** y **com.ms.com.NoAutoScripting** se omiten. Puede ser necesario modificar durante la actualización las aplicaciones de Visual J++ que dependen de la funcionalidad proporcionada por estas interfaces. Para obtener más información, vea [Problemas y soluciones: Java llama a COM](#).
- Actualmente, no existe compatibilidad para recibir eventos desencadenados por clases de Java o COM en clientes COM.
- No se permite el cálculo de referencias para VARIANT de tipo VT\_BYREF.

En objetos de Java expuestos a COM, el compilador de Visual J# (vjc.exe) omite las siguientes directivas @com:

- **@com.transaction**
- **@com.typeinfo**

En Visual J#, no se admite la exposición de controles WFC o Java Beans como controles ActiveX.

Tampoco se permiten monikers de Java en Visual J#.

## Vea también

[Escenarios no compatibles con Visual J#](#)

# Problemas conocidos y soluciones

En esta sección se comentan soluciones para problemas conocidos relacionados con la actualización:

- [Problemas y soluciones: Java llama a COM](#)
- [Problemas y soluciones: COM llama a Java](#)
- [Actualizar componentes de Java y COM que utilizan cálculo de referencias personalizado](#)

## Vea también

[Actualizar proyectos mediante la interoperabilidad de Java y COM | Escenarios no compatibles con Visual J#](#)

# Problemas y soluciones: Java llama a COM

En este tema se explica cómo resolver problemas que pueden encontrarse al llamar a un componente COM desde un cliente de Visual J#.

- Si el nombre de paquete de los contenedores JActiveX es el mismo que el del espacio de nombres en el que se generan los contenedores administrados, obtendrá un error de compilador.

Para resolverlo, hay que utilizar la opción `/namespace` cuando se generen contenedores administrados con la herramienta `Tlbimp.exe`. Para obtener más información sobre las opciones de `Tlbimp`, vea [Interoperar con código no administrado e Importador de la biblioteca de tipos \(Tlbimp.exe\)](#).

- En los casos en los que la aplicación de Java haga referencia a dos o más DLL COM que tengan interfaces o coclases COM con los mismos nombres y los mismos GUID, puede ser necesario resolver explícitamente el conflicto de nombres utilizando la opción `/jcpa` de `vjc.exe`. Las interfaces de Java (no marcadas con directivas `@com`) se pueden convertir en COM sólo si el lado COM espera una interfaz derivada de **IDispatch**. En otros casos, no se pueden convertir estas interfaces. Actualmente, la compatibilidad con el cálculo de referencias es completa sólo para interfaces COM generadas desde JActiveX.
- Las interfaces de marcador de los contenedores JactiveX como **com.ms.com.NoAutoMarshaling** y **com.ms.com.NoAutoScripting** se omiten. Puede ser necesario modificar durante la actualización las aplicaciones de Visual J++ que dependen de la funcionalidad proporcionada por estas interfaces, por los siguientes motivos:

Cuando la implementa una clase, la interfaz **com.ms.com.NoAutoScripting** impide que Microsoft Java Virtual Machine (MSJVM) exponga una implementación de **IDispatch** en un contenedor de la clase al que se pueda llamar mediante COM. Esto se puede utilizar para evitar que se llame a instancias de esta clase desde motores de secuencias de comandos como VBScript o Visual Basic. Puesto que Visual J# no admite esta interfaz, los contenedores a los que se puede llamar mediante COM de clases que implementan esta interfaz expondrán ahora una implementación de **IDispatch** después de la actualización.

En cuanto a la interfaz **com.ms.com.NoAutoMarshaling**, evita que MSJVM exponga el cálculo de referencias de subproceso libre en el contenedor de la clase al que se puede llamar mediante COM. Esto se puede utilizar al crear servidores locales COM donde se debe colocar el objeto de Java o COM en un apartamento concreto. Puesto que Visual J# no admite esta interfaz, los contenedores a los que se puede llamar mediante COM de clases que implementan esta interfaz expondrán ahora el cálculo de referencias de subprocesamiento libre después de la actualización.

- Actualmente, los HRESULT que no fallan, como `S_FALSE`, se tratan siempre como `S_OK` y no se inician excepciones de tipo **com.ms.com.ComSuccessException**.
- No se admite el cálculo de referencias de matrices. En la versión actual de Visual J#, sólo se calculan las referencias del primer elemento de la matriz para convertirlo de la clase de Java al cliente COM o viceversa.
- Es posible que los contenedores JActiveX que edita el desarrollador antes de la compilación no se compilen en `vjc.exe`. Si la compilación es correcta, hay todavía una posibilidad de problemas de incompatibilidad en tiempo de ejecución.
- Si el método de descriptor de acceso de una propiedad COM de una interfaz dispinterface no devuelve nada, entonces al intentar leer el valor de esa propiedad desde el cliente Java se obtendrá **ComFailException**. La solución es devolver siempre algún valor desde el servidor COM.
- Visual J# no admite los siguientes escenarios:
  - Convertir matrices seguras multidimensionales como parámetros en métodos COM. Esto dará lugar a errores en tiempo de ejecución.
  - Convertir parámetros de tipo `STRUCT**` donde `STRUCT` es una estructura COM definida en la biblioteca de tipos. En este caso, Visual J# da errores en tiempo de compilación.
- Un seguimiento de la pila durante un proceso de depuración puede no ser igual que en Visual J++ 6.0, puesto que la capacidad de interoperabilidad de Java y COM en Visual J# se sitúa en capas encima de la interoperabilidad COM de .NET Framework.

## Vea también

[Problemas conocidos y soluciones](#)

# Problemas y soluciones: COM llama a Java

Este tema explica cómo resolver problemas que se pueden encontrar al llamar a un componente de Visual J# desde un cliente COM.

- Cuando se intenta registrar una clase de Java de Visual J++ 6.0 que no está marcada con la directiva **@com.register** (por ejemplo, una clase que se registró utilizando la herramienta JavaReg.exe con la opción **/clsid**), se genera un CLSID automáticamente para esa clase. Esto puede dar lugar a que se interrumpan los clientes COM existentes. En estos casos, deberá asociar el atributo **@com.register** con la clase y especificar el CLSID que espera el cliente.
- En Visual J++ 6.0, se podía pasar a COM cualquier objeto de Java genérico como parámetro para un método. Los clientes COM pueden tener acceso a los métodos de ese objeto utilizando **IDispatch** o las interfaces COM específicas que implementa.

En Visual J#, no se puede convertir un objeto de Java a COM de este modo. El objeto se puede llamar desde COM sólo en los siguientes escenarios:

- Es una clase con el atributo **@com.register**.
- Implementa una interfaz COM generada con la herramienta JActiveX o las interfaces **com.ms.com**.

Si un objeto no reúne estas condiciones, todavía se le puede llamar desde COM, pero puede dar lugar a un comportamiento inesperado en tiempo de ejecución.

Esta limitación no se aplica si escribe código nuevo que utilice la semántica de interoperabilidad COM de .NET Framework y utilice sólo tipos de .NET Framework.

- Al actualizar componentes Java y COM de Visual J++ 6.0 que no se generaron con las clases de plantilla de JActiveX (**/c2j**) o no implementan interfaces generadas con JActiveX, debe asegurarse de que existe una biblioteca de tipos para el componente que se va a actualizar y que dicha biblioteca está registrada en el equipo. De lo contrario, el compilador de Visual J# emitirá la advertencia [VJS1553](#).

La solución consiste en llevar a cabo alguna de las siguientes acciones antes de la actualización a Visual J#:

- Utilizar la herramienta `vjreg.exe` que se distribuye con Visual J++ 6.0, generar y registrar una biblioteca de tipos para el componente Java que se va a exponer a COM, como sigue:

```
vjreg /typelib SomeTypeLib.tlb JavaComponent.class
```

O bien

Si existe también una biblioteca de tipos para el componente en el equipo pero no está registrada, utilice el siguiente código:

```
vjreg /regtypelib SomeTypeLib.tlb JavaComponent.class
```

- Los componentes administrados de Java registrados y expuestos a COM no pueden coexistir en el mismo equipo con sus homólogos de Visual J++ 6.0. Cuando se registra una clase de Java administrada como servidor COM, se registra con el mismo GUID utilizado por dicha clase en Visual J++ 6.0. Por tanto, las entradas de coclase se sobrescriben con `InProcServer32` como `mscorlib.dll`, en lugar de `msjava.dll`. En consecuencia, la clase no se puede utilizar más con Visual J++ 6.0. Los desarrolladores que deseen alojar la clase de Java en COM utilizando Visual J++ 6.0 y Visual J# no podrán hacerlo a menos que utilicen las herramientas de Visual J++ 6.0 o de Microsoft SDK para Java con el fin de registrar el archivo `.class`.
- Actualmente, los métodos o campos públicos que siguen el formato de nombre **getXXX/setXXX** de los componentes Java no se exponen a clientes COM como propiedades.
- Si la clase Java implementa una interfaz COM, sólo se puede tener acceso a los métodos que pertenecen a la interfaz COM utilizando la interfaz **IDispatch** desde COM. Los demás métodos públicos de la clase que no pertenecen a la interfaz COM implementada no están visibles para COM. Si la clase Java no implementa ninguna interfaz COM, se puede tener acceso a todos sus métodos públicos desde COM utilizando **IDispatch** (es decir, escenarios enlazados en tiempo de ejecución, como secuencias de comandos).

## Vea también

[Problemas conocidos y soluciones](#)

# Actualizar componentes de Java y COM que utilizan cálculo de referencias personalizado

Visual J++ 6.0 permite a los desarrolladores realizar cálculos personalizados de referencias de parámetros y estructuras de método durante la interoperabilidad de Java y COM. Cuando se utiliza la opción `/t:jnffile` o `/n:jnffile` al ejecutar la herramienta JActiveX en una biblioteca de tipos, los contenedores generados tienen información adicional necesaria para el cálculo personalizado de referencias. El archivo .jnf utilizado con la opción `/t` o `/n` especifica la clase del calculador personalizado de referencias que se debe utilizar para convertir un tipo específico de Java en su correspondiente tipo nativo. Se puede encontrar información adicional sobre el cálculo personalizado de referencias en Visual J++ 6.0 en la documentación de Microsoft Visual J++ 6.0 o Microsoft SDK para Java.

Microsoft Visual J# no admite actualmente el cálculo personalizado de referencias. Un intento de compilar contenedores JActiveX que utilicen calculadores personalizados de referencias para convertir los tipos de Java en tipos nativos dará lugar a un error en tiempo de compilación.

Common Language Runtime proporciona compatibilidad nativa con la interoperabilidad COM y el cálculo personalizado de referencias. Los usuarios pueden especificar una clase de calculador personalizado de referencias que se puede utilizar para realizar un cálculo personalizado de las referencias de tipos administrados para convertirlos en sus correspondientes representaciones nativas y viceversa. Para obtener información adicional sobre el cálculo personalizado de referencias en Common Language Runtime, vea [Cálculo personalizado de referencias](#).

Cuando se actualizan aplicaciones de Java o COM que utilizan cálculo de referencias personalizado, una forma posible de actualización consistiría en escribir de nuevo el componente para que utilice directamente los contenedores administrados generados por la herramienta Tlbimp.exe en lugar de utilizar los contenedores JActiveX. Common Language Runtime permite a los desarrolladores especificar clases de cálculo de referencias personalizado que calculen las referencias de argumentos de método durante la interoperabilidad COM. Al escribir dicho calculador de referencias personalizado, se puede convertir un tipo administrado escrito en Java en su representación nativa.

## Vea también

[Problemas conocidos y soluciones](#)

# Actualizar aplicaciones que cargan clases dinámicamente

En esta sección se tratan los siguientes temas:

- [Incompatibilidad con la variable CLASSPATH](#)
- [Enlaces en tiempo de compilación](#)
- [Enlaces en tiempo de ejecución: semántica de Class.forName](#)

## Vea también

[Compatibilidad con bibliotecas de clases](#)

# Incompatibilidad con la variable CLASSPATH

Visual J# no es compatible con la variable de entorno CLASSPATH. Esto afecta a:

- El modo en que las aplicaciones buscan y cargan clases enlazadas a la aplicación, tanto en tiempo de compilación como en tiempo de ejecución.
- El modo en que las aplicaciones buscan y cargan los archivos de recursos.

## **Vea también**

[Actualizar aplicaciones que cargan clases dinámicamente](#)



# Enlaces en tiempo de compilación

Para las clases enlazadas a una compilación en tiempo de compilación, los metadatos de la aplicación incluyen referencias a los ensamblados que contienen estas clases. En tiempo de ejecución, cuando se hace referencia a estas clases, Common Language Runtime busca y carga los ensamblados correspondientes utilizando su heurística de carga de ensamblados.

Para obtener más información sobre cómo busca y carga Common Language Runtime los ensamblados, vea [Cómo el motor de tiempo de ejecución ubica ensamblados](#).

## Vea también

[Actualizar aplicaciones que cargan clases dinámicamente](#)

# Enlaces en tiempo de ejecución: semántica de `Class.forName`

Las clases cargadas con la API **`Class.forName`** se enlazan dinámicamente con la aplicación en tiempo de ejecución.

El método **`Class.forName(String className)`** no utiliza la variable de entorno `CLASSPATH` para buscar clases. En su lugar, se buscan clases en las siguientes ubicaciones, por orden:

1. El ensamblado que llama.
2. Todos los ensamblados cargados actualmente en el dominio de la aplicación actual.
3. Ensamblados especificados en la etiqueta `<CODEBASE>` del archivo de configuración (`.config`) de la aplicación.
4. `ApplicationBase` de la aplicación.
5. Todas las DLL administradas encontradas en los directorios especificados por la etiqueta `<PROBING>` del archivo de configuración de la aplicación.

La búsqueda termina si se encuentra la clase en algún paso concreto. Si no se encuentra ningún archivo de configuración o si se producen errores al analizar el archivo, se omiten los pasos 3 a 5. Si no se encuentra la clase en ninguno de los pasos, se inicia **`ClassNotFoundException`**.

Visual J# proporciona también un método adicional en la clase **`java.lang.Class`** para buscar y cargar clases de un ensamblado de la siguiente forma:

```
Class.forName(String assemblyName, String className, boolean absolutePath)
```

Este método busca la clase especificada por el parámetro *className* en el ensamblado especificado por el parámetro *assemblyName*. El parámetro *assemblyName* debe especificar el nombre del ensamblado que se va a mostrar (vea [AssemblyName \(Clase\)](#) para obtener más información). Si el parámetro booleano *absolutePath* es verdadero, *assemblyName* representa una ruta absoluta que especifica la ubicación del ensamblado. Si *absolutePath* es falso, se supone que no se ha especificado ninguna ruta y se utiliza la heurística de búsqueda de ensamblados de Common Language Runtime para buscar y cargar el ensamblado. Si no se encuentra la clase en el ensamblado especificado o no se encuentra dicho ensamblado, se inicia **`ClassNotFoundException`**.

Para obtener más información, vea [Caché de ensamblados global](#), [Ensamblados](#), [Dominios de aplicación](#) y [Cómo el motor de tiempo de ejecución ubica ensamblados](#).

## Vea también

[Actualizar aplicaciones que cargan clases dinámicamente](#)

# Actualizar aplicaciones que utilizan cargadores de clases personalizados

No hay compatibilidad con **ClassLoader** para convertir código de bytes en un objeto **Class**.

Los siguientes métodos no son compatibles:

- **ClassLoader.defineClass**
- **ClassLoader.resolveClass**

Estos métodos inician **com.ms.vjsharp.MethodNotSupportedException** en Visual J#.

Cargadores de clases personalizados amplían **java.lang.ClassLoader** y reemplazan al método **loadClass** para su implementación específica. Si estas implementaciones de **loadClass** llaman a alguno de los métodos **ClassLoader** anteriores o a ambos, se deben modificar para cargar ensamblados administrados y buscar después las clases especificadas en ellos.

Puesto que **ClassLoader.defineClass** y **ClassLoader.resolveClass** ya no son compatibles, **Class.getClassLoader** devuelve siempre null, incluso cuando se utiliza un cargador de clases personalizado.

No se recomiendan los cargadores de clases personalizados para desarrolladores que utilizan Visual J#. En su lugar, utilice las API de .NET Framework para buscar y cargar clases de ensamblados del Lenguaje intermedio de Microsoft (MSIL).

## Vea también

[Actualizar aplicaciones que cargan clases dinámicamente](#)

# Actualizar aplicaciones que utilizan la semántica de seguridad de Visual J++ 6.0

Las aplicaciones escritas en Visual J# siguen la semántica de seguridad de .NET Framework y Common Language Runtime. Esto se aplica a las aplicaciones nuevas escritas con Visual J# orientadas a bibliotecas de .NET Framework, así como a aplicaciones de Visual J++ 6.0 actualizadas a Visual J#.

## Administradores de seguridad personalizados en aplicaciones de Visual J#

Visual J# no admite administradores de seguridad personalizados. Por tanto, no se admiten los siguientes métodos de las clases **java.lang.System** y **java.lang.SecurityManager**, que inician la excepción **com.ms.vjsharp.MethodNotSupportedException**:

- **System.setSecurityManager**
- **SecurityManager.classDepth**
- **SecurityManager.classLoaderDepth**
- **SecurityManager.currentClassLoader**
- **SecurityManager.currentLoadedClass**
- **SecurityManager.getClassContext**
- **SecurityManager.inClass**
- **SecurityManager.inClassLoader**
- El método **System.getSecurityManager** devolverá siempre null.

Al actualizar una aplicación de Visual J++ 6.0 existente que implemente un administrador de seguridad personalizado y utilice los métodos anteriores para implementar una directiva de seguridad, se debe modificar la aplicación para utilizar la semántica de seguridad y las clases proporcionadas por .NET Framework.

Esto suele significar el uso de API equivalentes en el espacio de nombres **System.Security** y modificar la directiva de seguridad de .NET Framework en el equipo para activar las restricciones necesarias.

## Tener acceso a clases de paquetes JDK nivel 1.1.4, WFC y otros paquetes com.ms.\*

En Visual J#, sólo las aplicaciones que son totalmente de confianza pueden tener acceso a las clases de los paquetes JDK nivel 1.1.4, WFC y com.ms.\*. Esto significa que sólo las aplicaciones que se ejecutan en el grupo de código Mi PC o un grupo de código al que la directiva de seguridad de .NET Framework ha concedido el conjunto de permisos FullTrust pueden tener acceso a estas clases. Si un llamador de confianza parcial intenta tener acceso a estas clases, se genera una excepción de seguridad.

Sin embargo, Visual J# comprueba únicamente los llamadores inmediatos de estas clases. Por tanto, los desarrolladores podrán generar bibliotecas totalmente de confianza y seguras, y exponerlas a código de confianza parcial. Los desarrolladores de estas bibliotecas deben asegurarse de que sus bibliotecas son totalmente seguras y no dejan el equipo expuesto a ataques.

Las clases de JDK nivel 1.1.4 de Visual J# están diseñadas para demandar los permisos apropiados de .NET Framework antes de realizar operaciones delicadas. Por tanto, los llamadores de total confianza de estas clases podrán activar la semántica de Deny y PermitOnly. Asimismo, cuando llamadores de segundo nivel (potencialmente maliciosos) llaman a estos llamadores de confianza, se permite un cierto grado de protección (estos llamadores de segundo nivel no podrán realizar operaciones para las que no tienen permisos).

Sin embargo, a diferencia de las clases de JDK nivel 1.1.4, las clases de paquetes WFC y com.ms.\* no llevan a cabo estas solicitudes de permisos de .NET Framework. Por tanto, los llamadores de total confianza de estos llamadores no podrán implementar la semántica de Deny y PermitOnly. Determinadas clases, como E/S de archivos, realizarán un recorrido de la pila completo en busca de un conjunto de permisos no restringidos y son una excepción.

Para obtener más información sobre el modelo de seguridad de .NET Framework y la modificación de las directivas de seguridad de .NET Framework, vea [Seguridad en .NET Framework](#).

## Vea también

[Compatibilidad con bibliotecas de clases](#) | [Actualizar aplicaciones que utilizan proveedores de seguridad de otros fabricantes](#)

# Actualizar aplicaciones que utilizan proveedores de seguridad de otros fabricantes

Visual J# permite a los usuarios agregar proveedores de seguridad de otros fabricantes que se pueden utilizar para operaciones como la generación de claves y firmas. Los usuarios pueden especificar, a través de un archivo de configuración de seguridad, los proveedores de seguridad instalados en el sistema y el orden en el que se buscará la implementación de un algoritmo en ellos.

Vea [SecurityProviders \(Ejemplo\)](#) para ver un ejemplo de cómo utilizar un archivo de configuración de seguridad.

Los usuarios de Visual J# pueden utilizar el archivo de configuración `vjsharp.config` del directorio `%windir%\Microsoft .NET\Framework\v<%version%>` para especificar esta configuración. Es un archivo XML y toda la configuración de seguridad está especificada dentro de la etiqueta `<security>`. El formato típico de este archivo es:

```
<vjsharpconfiguration>
  <security>
    <packageinfo>
      <description>Description of the package</description>
      <loadinfo class="packageName.className, assemblyName, Version=assemblyVersion, Cu
lture=assemblyCulture, PublicKeyToken=assemblyPublicKeyToken">
        <load/> [|| <noload/>]
      </loadinfo>
    </packageinfo>
    .
    .
    .
  </security>
</vjsharpconfiguration>
```

Las etiquetas del archivo se utilizan del siguiente modo:

`<packageinfo>`

Especifica todos los proveedores de seguridad instalados.

`<description>`

Permite al usuario proporcionar una descripción del paquete y no juega ningún papel en la carga de proveedores.

`<loadinfo>`

Especifica el nombre de la clase de proveedor del ensamblado y el paquete de proveedor de seguridad donde se encuentra. Para los ensamblados instalados en la Caché de ensamblados global, el nombre del ensamblado debería también incluir la versión, la referencia cultural y el símbolo de clave pública. Si sólo se especifica el nombre de ensamblado, sólo se utiliza `ApplicationBase` para buscar el ensamblado. Para los ensamblados no instalados en la Caché de ensamblados global también puede especificar el nombre completo de la ruta de acceso al ensamblado. En tal caso, el ensamblado específico se carga desde el directorio indicado.

`<load> 0 <noload>`

Especifican si se debería cargar u omitir el ensamblado para la búsqueda actual. Si se especifica `<noload>`, se omite el ensamblado.

El archivo `jsharp.config` no se crea de manera predeterminada durante la instalación. Cuando no hay un archivo de configuración, se utiliza el proveedor de seguridad predeterminado.

Para usar otro proveedor debe crear este archivo en el directorio `%windir%\Microsoft \Framework\v<Version>`. El formato del archivo debe ser el especificado anteriormente. Los proveedores se cargan en el orden en el que aparecen en la etiqueta `<security>`. Los errores encontrados en el formato de una etiqueta `<packageinfo>` se omiten en modo desatendido. Los ensamblados de los proveedores enumerados en este archivo deben instalarse en un lugar donde los pueda encontrar la heurística de búsqueda y carga de ensamblados de Common Language Runtime de .NET Framework. Cuando se especifica el archivo `vjsharp.config`, no se busca el proveedor predeterminado a menos que se especifique en el archivo. Una etiqueta `<loadinfo>` para el proveedor predeterminado puede tener el siguiente aspecto:

```
<loadinfo class="com.ms.vjsharp.security.provider.ms, vjslib, Version=1.0.4030.0, Culture=neu
tral, PublicKeyToken=B80725BFD0434260">
  <load/>
</loadinfo>
```

**Vea también**



# Actualizar paquetes com.ms.\* a bibliotecas de clases de .NET Framework

Visual J# admite paquetes com.ms.\* que son críticos para actualizar correctamente aplicaciones de Visual J++ 6.0 a Visual J#. La lista completa de paquetes está disponible en [Bibliotecas de clases compatibles](#).

La biblioteca de clases de .NET Framework proporciona funcionalidad equivalente a muchos de los paquetes com.ms.\* no compatibles. Muchas de estas clases com.ms.\* tienen equivalentes directos en la biblioteca de clases de .NET Framework.

La tabla siguiente muestra los equivalentes en .NET Framework de algunos de los paquetes com.ms.\* más utilizados.

Paquete com.ms.*	Biblioteca de clases de .NET Framework
com.ms.mtx	System.EnterpriseServices
com.ms.asp	System.Web
com.ms.iis.asp	System.Web
com.ms.util.xml	System.Xml
com.ms.xml.om	System.Xml
com.ms.xml.parser	System.Xml
com.ms.xml.util	System.Xml
com.ms.security	System.Security y System.Security.Policy
com.ms.security.permissions	System.Security.Permissions
com.ms.vm	System.WeakReference

También se proporcionan instrucciones de actualización para pasar de muchos de los paquetes com.ms.\* enumerados a los correspondientes espacios de nombres en .NET Framework.

## En esta sección

[Actualizar componentes que utilizan el paquete com.ms.xml.\\*](#)

Muestra cómo actualizar una aplicación que lee un archivo XML sencillo utilizando las clases de **System.Xml**.

[Actualizar componentes que utilizan el paquete com.ms.mtx](#)

Muestra cómo actualizar una aplicación que utiliza el paquete **com.ms.mtx** utilizando las clases de **System.EnterpriseServices**.

[Actualizar componentes que utilizan el paquete com.ms.wfc.html](#)

Muestra cómo actualizar la etiqueta <OBJECT> en el código del **com.ms.wfc.html**.

## Secciones relacionadas

[Actualizar proyectos de Visual J++ 6.0](#)

Proporciona recomendaciones y procedimientos para actualizar proyectos de Visual J++ 6.0 a Visual J#.

# Actualizar componentes que utilizan el paquete com.ms.xml.\*

Las clases de los paquetes com.ms.xml.\* se pueden utilizar para cargar, leer, modificar y guardar documentos XML. En la documentación de Visual J++ 6.0 y Microsoft SDK para Java, se tratan estas clases con detalle. Visual J# no admite estos paquetes. Al actualizar una aplicación de Visual J++ que utiliza estos paquetes, se recomienda pasar a las clases correspondientes en el espacio de nombres **System.Xml** de .NET Framework.

Las clases de **System.Xml** proporcionan funcionalidad equivalente a los paquetes com.ms.xml.\* y cumplen los estándares W3C. En los paquetes com.ms.xml.\*, las clases **Document**, **ElementCollection**, **ElementImpl** y **Element** se encuentran entre las más utilizadas. Las clases **XMLDocument**, **XMLNodeList** y **XMLNode** de **System.Xml** son equivalentes directos.

## Ejemplo

El ejemplo siguiente muestra cómo actualizar a Visual J# una aplicación de Visual J++ que lee un archivo XML sencillo utilizando las clases de **System.Xml**.

### SampleFile.Xml

```
<?xml version="1.0" encoding="utf-8" ?>
<bookdetails>
  <publisher>Microsoft Press</publisher>
  <book>
    <name>Designing Component-Based Applications</name>
    <price>12.45</price>
  </book>
  <book>
    <name>Professional Visual C++ MTS Programming</name>
    <price>21.90</price>
  </book>
  <book>
    <name>Essential COM+, 2nd Edition</name>
    <price>33.10</price>
  </book>
</bookdetails>
```

### Aplicación de Visual J++ 6.0

```
import com.ms.xml.om.*;
public class XMLUpgrade
{
    public static void main(String[] args)
    {
        try
        {
            // Create an XML document object and load an XML file
            Document xmlDoc = new Document();
            xmlDoc.load("SampleFile.xml");

            // Get the root node of the document
            Element xmlElement = xmlDoc.getRoot();
            // Get a collection of child nodes under the root node
            ElementCollection xmlEleCo1 = xmlElement.getChildren();
            // Get all nodes named 'book' in the collection
            ElementCollection xmlEleCo2 = (ElementCollection)
            xmlEleCo1.item("book");
            int length = xmlEleCo2.getLength();
            for (int i=0; i < length; i++)
            {
                // Retrieve the current element (book) from the collection
                Element xmlElement1 = xmlEleCo2.getChild(i);
                // Print the non-marked up text of the first child
                // (book name)
                System.out.println(xmlElement1.getChild(1).getText());
            }
        }
    }
}
```



```

        catch(Exception e)
        {
            System.out.println("An exception occurred");
        }
    }
}

```

## Después de la actualización a Visual J#

```

import System.Xml.*;
public class XMLUpgrade
{
    public static void main(String [] args)
    {
        try
        {
            // Create an XML document object and load an XML file
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load("SampleFile.xml ");

            // Get all nodes named 'book' in the document
            XmlNodeList xnl = xd.GetElementsByTagName("book");
            int length = xnl.get_Count();

            XmlNode xn = null;
            for (int i=0; i < length; i++)
            {
                // Retrieve the current element (book) from the collection
                xn = xnl.Item(i);

                // Print the non-marked-up text of the first child
                // (book name)
                System.Console.WriteLine(xn.get_FirstChild().get_InnerText());
            }
        }
        catch (Exception e)
        {
            System.Console.WriteLine("An exception occurred");
        }
    }
}

```

**Nota** Los cambios en el código fuente se deben realizar manualmente. El Asistente para actualización no agregará automáticamente las clases de .NET Framework a la aplicación.

## Vea también

[Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework](#)

# Actualizar componentes que utilizan el paquete com.ms.mtx

Las clases del paquete **com.ms.mtx** permiten que esté disponible la funcionalidad de Microsoft Transaction Server (MTS) para los desarrolladores de Java. En la documentación de Visual J++ 6.0 y Microsoft SDK para Java, se tratan estas clases con detalle. Visual J# no admite el paquete **com.ms.mtx**. Al actualizar aplicaciones de Visual J++ que utilizan estos paquetes, se recomienda pasar a las clases correspondientes en el espacio de nombres **System.EnterpriseServices** de .NET Framework. Las clases de **System.EnterpriseServices** proporcionan funcionalidad equivalente a las clases del paquete **com.ms.mtx**.

A continuación, se muestra la forma canónica de escribir componentes MTS en Visual J++ utilizando el paquete **com.ms.mtx**:

```
public int transactMethod ()
{
    IObjectContext context;
    boolean success = false;
    IMtxComp mtxCmp = null;

    // Obtain MTS context
    context = (IObjectContext) Mtx.GetObjectContext();

    // Obtain resources. For example, database connections
    // Use resources
    // Perform one piece of work for one client
    // Invoke other MTS components to do some of the work
    mtxCmp = (IMtxComp) context.CreateInstance(CMtxComp.clsid, IMtxComp.iid);

    // all went well
    success = true;
    return 0;

    finally
    {
        // release resources
        if (success)
            context.SetComplete();
        else
            context.SetAbort();
    }
}
```

Algunas de las clases más utilizadas son **com.ms.mtx.IObjectContext**, **com.ms.mtx.IGetContextProperties**, **com.ms.mtx.Context** y **com.ms.mtx.Mtx**. La clase **ContextUtil** del espacio de nombres **System.EnterpriseServices** es un equivalente directo de las clases **com.ms.mtx.IObjectContext** y **com.ms.mtx.Mtx**. Las demás clases del paquete **com.ms.mtx**, como **ISharedProperty**, **ISharedPropertyGroup** y **ISharedPropertyGroupManager** tienen también equivalentes directos en el espacio de nombres **System.EnterpriseServices** de .NET Framework.

## Ejemplo

El ejemplo siguiente muestra cómo actualizar una aplicación de Visual J++ a Visual J# utilizando las clases de **System.EnterpriseServices**.

### Aplicación de Visual J++ 6.0

```
import com.ms.mtx.*;

/** @com.transaction(required) */
/** @com.register(clsid=E96931CC-DFDC-497C-B695-F9EC12F8F470) */
public class MtxServer
{
    public String transactMethod(String name)
    {
        String greeting = null;
        boolean complete = false;

        try
```

```

    {
        // This would probably be a database write in
        // real world application code
        greeting = "Hello, " + name;
        complete = true;
    }

    finally
    {
        IObjectContext ctx = MTx.GetObjectContext();
        if (complete)
        {
            ctx.SetComplete();
        }
        else
        {
            ctx.SetAbort();
        }
    }
    return greeting;
}
}

```

### Después de la actualización a Visual J#

El código siguiente muestra cómo lograr funcionalidad equivalente utilizando las clases del espacio de nombres

**System.EnterpriseServices** de .NET Framework. Los métodos de las interfaces públicas del componente actualizado deben utilizar tipos de .NET Framework en lugar de tipos de Java.

```

import System.EnterpriseServices.*;
import System.Runtime.InteropServices.*;

/** @attribute GuidAttribute("E96931CC-DFDC-497C-B695-F9EC12F8F470") */
/** @attribute TransactionAttribute(TransactionOption.Required) */
// The class has to extend System.EnterpriseServices.ServicedComponent
public class MtxServer extends ServicedComponent
{
    public String transactMethod(String name)
    {
        String greeting = null;
        boolean complete = false;

        try
        {
            // This would probably be a database write in
            // real world application code
            greeting = "Hello, " + name;
            complete = true;
        }
        finally
        {
            if (complete)
            {
                ContextUtil.SetComplete();
            }
            else
            {
                ContextUtil.SetAbort();
            }
        }
        return greeting;
    }
}

```

En algunos casos, se utiliza el paquete **com.ms.mtx** en componentes que utilizan el paquete **com.ms.asp** para recuperar el objeto context. En estos casos, se puede utilizar la clase **System.Web.HttpContext** de .NET Framework para recuperar el objeto **Request** o **Response**.

## **Vea también**

[Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework](#)

# Actualizar componentes que utilizan el paquete com.ms.wfc.html

Visual J# es totalmente compatible con el paquete **com.ms.wfc.html**. Sin embargo, los parámetros de la etiqueta <OBJECT> han cambiado para permitir la carga de las clases de ensamblados administrados teniendo en cuenta la versión.

Por tanto, cuando se actualizan componentes de Visual J++ 6.0 que utilizan el paquete **com.ms.wfc.html**, se deben modificar en consecuencia las páginas HTML que contienen la etiqueta <OBJECT>.

La etiqueta <OBJECT> que se utiliza en Visual J++ 6.0 tiene el siguiente aspecto:

```
<OBJECT CLASSID="java:com.ms.wfc.html.DhModule" height=0 width=0 ... VIEWASTEXT>
<PARAM NAME=__CODECLASS VALUE=UserClass>
<PARAM NAME=CABBASE VALUE=UserCabFile.CAB>
</OBJECT>
```

La etiqueta <OBJECT> que se debe utilizar en Visual J# tiene el siguiente aspecto:

```
<OBJECT CLASSID="CLSID:CF83EA5E-0FFD-4476-9BF7-6C15F7F8BDCF" height=0 width=0 ... VIEWASTEXT>
  <PARAM NAME=CODECLASS VALUE=UserAssembly#UserClass>
</OBJECT>
```

donde:

## CLASSID

Hace referencia al control que utiliza Visual J# para alojar la clase definida por el usuario. Los desarrolladores no deben cambiar este valor.

## CODECLASS

Hace referencia a la clase definida por el usuario (que extiende **DhDocument**) y al ensamblado administrado que contiene esta clase. El símbolo '#' separa el nombre de ensamblado y el nombre de clase. El nombre de ensamblado debe ser el nombre del archivo y debe incluir la extensión del archivo. Se intentará cargar el ensamblado desde la misma ubicación donde se encuentra la página HTML que tiene acceso a él. También es posible especificar una ruta relativa para el ensamblado.

**Nota** La clase definida por el usuario siempre se aloja en la versión más reciente de Visual J# disponible en el equipo.

En esta versión de Visual J# no se admite la etiqueta de parámetro `VJSVERSION`, por lo que se pasará por alto si se incluye en una etiqueta <OBJECT>.

Por ejemplo, una etiqueta <OBJECT> típica utilizada en Visual J# puede tener el siguiente aspecto:

```
<OBJECT CLASSID="CLSID:CF83EA5E-0FFD-4476-9BF7-6C15F7F8BDCF" height=0 width=0 ... VIEWASTEXT>
  <PARAM NAME=CODECLASS VALUE=MyAssembly.dll#MyUserClass>
</OBJECT>
```

Visual J# cargará y ejecutará un componente sólo si es de total confianza para la directiva de seguridad de .NET Framework del equipo cliente. Para ser considerado totalmente de confianza por la directiva de seguridad del cliente, se debe dar alguno de los siguientes casos:

- El componente debe estar firmado con un par de claves y se debe haber modificado la directiva de seguridad de .NET Framework en el equipo cliente para conceder el conjunto de permisos denominado FullTrust a todos los componentes o aplicaciones firmados con este par de claves.
- El componente debe estar firmado con un certificado Authenticode y se debe haber modificado la directiva de seguridad de .NET Framework en el equipo cliente para conceder el conjunto de permisos con nombre FullTrust a todos los componentes o aplicaciones firmados con este certificado.
- Se ha modificado la directiva de seguridad de .NET Framework en el equipo cliente para conceder el conjunto de permisos denominado FullTrust a controles descargados desde el sitio Web donde se aloja el componente.

Un componente que el equipo cliente no considere de total confianza no se podrá ejecutar. Si se carga una página que contiene un componente de este tipo en Internet Explorer, se muestra un mensaje de error en la barra de estado. En función de la configuración de seguridad de Internet Explorer en el cliente, la carga de una página con un componente de este tipo puede dar

lugar a que aparezca un cuadro de diálogo de advertencia de ActiveX cada vez que se cargue la página. El usuario debe hacer clic en Aceptar en el cuadro de diálogo para proceder con la carga del componente.

### **Vea también**

[Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework](#)

# Problemas comunes al actualizar proyectos de Visual J++ 6.0

En general, debe asegurarse de que un proyecto de Visual J++ 6.0 se genera y ejecuta en ese entorno antes de intentar actualizarlo a Visual J#. Si un proyecto devuelve errores al generarlo en Visual J++ 6.0, es muy probable que el análisis de actualización sea incompleto y no detecte algunos o ninguno de los problemas de actualización.

Hay una serie de diferencias entre los proyectos de Visual J# y los de Visual J++ 6.0. A continuación, se exponen algunos de los problemas relacionados con la actualización de proyectos:

- CLASSPATH es una característica de Visual J++ 6.0 que permite al usuario especificar las ubicaciones donde buscar referencias adicionales. Este valor no se admite en Visual J#. Para agregar referencias adicionales, cree una biblioteca de los archivos de CLASSPATH y agréguela como una referencia al proyecto de Visual J#. Vea [Agregar y quitar referencias de proyecto](#) para obtener más información. Vea [Conversor binario de Visual J#](#) para obtener más información sobre la conversión de bibliotecas y aplicaciones disponibles sólo como archivos de código de bytes (.class) en ensamblados MSIL.
- El empaquetado de resultados se realiza ahora utilizando [proyectos de instalación](#) en Visual Studio .NET. La información de formato de empaquetado, como el empaquetado del proyecto como un archivo CAB, ZIP o ejecutable autoextraíble, no se actualiza al nuevo proyecto.
- La información de implementación, como el nombre de la compañía y la versión del producto, que se almacena en el ejecutable empaquetado, se actualiza como propiedades de ensamblado en Visual J#. Vea [Establecer atributos de ensamblado](#) para obtener más información sobre los atributos de ensamblado y cómo utilizarlos.
- La configuración anterior y posterior a la generación no se admite en Visual J#. Estas configuraciones del proyecto de Visual J++ 6.0 no se actualizan a Visual J#. Si desea crear un paso de generación personalizada en un proyecto de Visual J#, agregue un proyecto de archivo MAKE de C++ a la solución (vea [Crear un proyecto de archivos MAKE](#)) y cambie la propiedad **Tipo de configuración a Utilidad** (vea [Página de propiedades General](#) para obtener más información). Vea [Introducción a los pasos de generación personalizada y los eventos de generación](#) para obtener más información.
- Estas configuraciones de la solución de proyectos de Visual J++ 6.0 no se actualizan al archivo de solución de Visual J#. El archivo de solución y el archivo de proyecto (.sln y .vjsproj) nuevos de Visual J# se deben agregar al control de código fuente. Utilice el comando Cambiar control de código fuente del menú Control de código fuente para volver a enlazar los proyectos con sus ubicaciones de servidor.
- En Visual J++ 6.0, un proyecto puede tener acceso a las clases definidas en otro proyecto. Si se abre este proyecto en el entorno de desarrollo de Visual J#, no se compilará. Debe agregar una referencia de proyecto, como se indica a continuación:
  1. Abra la solución de Visual J++ 6.0 en Visual J# y, cuando se le pida, actualice cada proyecto de la solución.
  2. Si es necesario, agregue una referencia a vjswfc.dll y vjswfhtml.dll. Vea [Agregar referencia \(Cuadro de diálogo\)](#) para obtener más información.
  3. Asegúrese de que la propiedad **Tipo de resultados** del proyecto a cuya clase se va a hacer referencia es un proyecto de **Biblioteca de clases**. Para obtener más información, vea [General, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#). Genere el proyecto.
  4. Cree una referencia al proyecto que contiene la clase del proyecto que utiliza la clase.
- En una solución de Visual J++ 6.0 que contiene varios proyectos, si no se actualizan uno o más proyectos la primera vez que se actualiza la solución, los demás proyectos se marcan como no generables. La próxima vez que abra la solución actualizada, se le pedirá que actualice el resto de los proyectos. Incluso después de actualizarlos, continuarán marcados como no generables en la solución. Esto significa que, cuando se genere la solución, se omitirán estos proyectos.
 

Para incluirlos en la generación de la solución, proceda como se indica a continuación:

  1. Haga clic con el botón secundario del *mouse* (ratón) en el nodo de la solución en el Explorador de soluciones y elija **Propiedades**.
  2. En el cuadro de diálogo **Páginas de propiedades de la solución**, seleccione la carpeta **Configuración** dentro de la carpeta **Propiedades de configuración**.
  3. En el cuadro combinado **Configuración** de la parte superior del cuadro de diálogo, elija **Todas las configuraciones**.
  4. Para cada proyecto, asegúrese de que está activada la casilla de verificación de la columna **Generar**.
- Una solución de Visual J++ 6.0 que tenga varios proyectos y una o más dependencias circulares no se puede actualizar a Visual J#. Por ejemplo, si una solución de Visual J++ 6.0 contiene los proyectos A y B, A puede utilizar las clases de B y B

puede utilizar las clases de A si se define la propiedad **Merge all Project-specific Classpaths in solution** (está definida de manera predeterminada para todos los proyectos).

En Visual J#, esta solución se actualizaría de manera que el proyecto A agregue una referencia al proyecto B y éste agregue una referencia al proyecto A, siempre y cuando ambos proyectos sean de bibliotecas de clases. Sin embargo, no se permiten referencias circulares en Visual J#. Por tanto, debe combinar los dos proyectos en un único ensamblado en Visual J#.

## **Vea también**

[Actualizar proyectos de Visual J++ 6.0](#)



# Trabajar con Visual J++ 6.0 y Visual J#

Tras actualizar la solución de Visual J++ 6.0, puede trabajar en Visual J++ 6.0 y Visual J# conjuntamente. La solución original de Visual J++ 6.0 se puede abrir y ejecutar en Visual J++ 6.0, mientras que la solución actualizada se puede abrir y ejecutar en Visual J#. Esto permite probar las distintas características de ambos entornos.

El nuevo archivo de solución de Visual J# (.sln) no se puede abrir en Visual J++ 6.0.

Después de la actualización, se agregará el informe generado por el Asistente para actualización al proyecto de Visual J++ 6.0, puesto que el informe se genera en el mismo directorio que el proyecto original. Esto no afectará a la generación y ejecución del proyecto de Visual J++ 6.0, porque el informe de actualización es un archivo HTML.

Puesto que los archivos de código fuente se comparten entre los proyectos original y actualizado, los cambios realizados en estos archivos en un proyecto afectarán al otro proyecto.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#)

# Proyectos de Visual J#

Con una excepción, los tipos de proyecto disponibles en el sistema de proyectos de Visual J# son los mismos que los tipos disponibles en Visual Basic y Visual C#. Actualmente, Visual J# no es uno de los lenguajes disponibles para crear proyectos de extensibilidad (proyectos de complementos) mediante el Asistente para complementos. Para obtener más información, vea [Crear complementos](#).

Para obtener más información sobre los tipos de proyecto, vea [Proyectos de Visual Basic y Visual C#](#). Toda la documentación sobre la creación, administración y desarrollo de proyectos incluida en [Visual Basic y Visual C#](#) se aplica a Visual J# también.

## En esta sección

### [Establecer las propiedades de un proyecto de Visual J#](#)

Proporciona instrucciones para tener acceso a propiedades de proyecto.

### [Modelo de objetos de extensibilidad de Visual J# para proyectos](#)

Proporciona información de referencia para manipular proyectos mediante programación.

### [Utilizar recursos en aplicaciones de Visual J#](#)

Explica la semántica para buscar y cargar recursos en aplicaciones de Visual J#.

## Secciones relacionadas

### [Propiedades de un proyecto de Visual J#](#)

Proporciona vínculos a temas de referencia sobre las propiedades de los proyectos de Visual J#.

### [Actualizar proyectos de Visual J++ 6.0](#)

Proporciona recomendaciones y procedimientos para actualizar proyectos de Visual J++ 6.0 a Visual J#.

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

### [Proyectos de Visual Basic y Visual C#](#)

Proporciona una lista de los tipos de proyecto que se pueden crear con plantillas de proyecto.

# Establecer las propiedades de un proyecto de Visual J#

Las propiedades de un proyecto especifican la forma de generarlo y depurarlo.

## Acceso a Propiedades del proyecto

Las páginas de propiedades de un proyecto de Visual J# sólo están disponibles cuando el proyecto está seleccionado en el Explorador de soluciones, mientras que no están disponibles para archivos individuales. Sin embargo, se puede tener acceso a las propiedades de un archivo en la ventana **Propiedades**.

### Para obtener acceso a las propiedades del proyecto

1. Con un proyecto seleccionado en el **Explorador de soluciones**, haga clic en **Páginas de propiedades** en el menú **Ver**.  
Observe las dos carpetas del cuadro de diálogo **Páginas de propiedades** de un proyecto de Visual J#: **Propiedades comunes** y **Propiedades de configuración**. Las propiedades de **Propiedades de configuración** sólo se pueden establecer mediante la configuración del proyecto.
2. En la lista desplegable **Configuración** y con la carpeta **Propiedades de configuración** seleccionada, elija la configuración para la que desea establecer propiedades del proyecto. Si ha seleccionado la carpeta **Propiedades comunes**, la lista desplegable **Configuración** no estará disponible.
3. En el panel izquierdo, seleccione una carpeta y una página de propiedades.
4. En el panel derecho, establezca las propiedades disponibles.
5. Haga clic en **Aceptar** para guardar la configuración. Si desea que un cambio surta efecto inmediatamente, haga clic en **Aplicar**.

**Nota** Las propiedades que no están disponibles se definen al crear el proyecto; se muestran sólo a título informativo.

Las propiedades se guardan en el archivo .vjsproj del proyecto.

Para obtener información acerca del menú **Generar** en el entorno de Visual Studio®, vea [Generaciones predeterminadas y personalizadas](#).

## Pasos de generación personalizada

Observe que un proyecto de Visual J# no admite pasos de generación personalizada. Si desea crear un paso de generación personalizada en un proyecto de Visual J#, agregue un proyecto de archivo MAKE de C++ a la solución (vea [Crear un proyecto de archivos MAKE](#)) y cambie el **Tipo de configuración** a Utilidad (vea [Página de propiedades General](#) para obtener más información). Vea [Introducción a los pasos de generación personalizada y los eventos de generación](#) para obtener más información.

## Obtener ayuda sobre propiedades

Hay documentación disponible para cada propiedad; para ello, haga clic en el botón **Ayuda**.

- [General, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Configuración del Web, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Valores predeterminados del diseñador, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Ruta de acceso de referencias, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Generar eventos, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Generar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Depurar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)
- [Avanzadas, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto>](#) (Cuadro de diálogo)

## Vea también

[Proyectos de Visual J#](#) | [Propiedades de un proyecto de Visual J#](#) | [Opciones del compilador de Visual J#](#) | [Propiedades del proyecto](#) | [Propiedades \(Ventana\)](#)

# Modelo de objetos de extensibilidad de Visual J# para proyectos

Los usuarios de Visual J# pueden manipular proyectos mediante programación utilizando las propiedades, métodos y eventos descritos en [Modelo de objetos de extensibilidad de Visual Basic y Visual C# para proyectos](#), con las siguientes instrucciones adicionales.

## Propiedades de objeto de configuración

La siguiente propiedad de objeto de configuración es compatible con proyectos de Visual J#:

### AdditionalOptions

Especifica o devuelve, como una cadena, las opciones adicionales que se pasarán al compilador. Esto puede resultar útil si una opción del compilador no se expone de otro modo como una propiedad en el modelo de objetos. Para obtener más información, vea [AdditionalOptions](#).

Las siguientes propiedades de objeto de configuración no son aplicables a proyectos de Visual J#:

- [AllowUnsafeBlocks](#)
- [CheckForOverflowUnderflow](#)
- [DefineDebug](#)
- [DefineTrace](#)
- [DocumentationFile](#)
- [FileAlignment](#)
- [IncrementalBuild](#)
- [RemoveIntegerChecks](#)

## Propiedades de objeto de proyecto

Las siguientes propiedades de objeto de proyecto no son aplicables a proyectos de Visual J#:

- [ApplicationIcon](#)
- [OptionCompare](#)
- [OptionExplicit](#)
- [OptionStrict](#)

## Vea también

[Proyectos de Visual J#](#) | [Modelo de objetos de extensibilidad de Visual Basic y Visual C# para proyectos](#)

# Utilizar recursos en aplicaciones de Visual J#

Visual J# no admite CLASSPATH. Las aplicaciones que se ejecutan en Visual J# utilizan la semántica de .NET Framework para buscar y cargar recursos. Para las aplicaciones nuevas escritas en Visual J#, se deben especificar archivos de recursos con el formato de archivo .resources y, a continuación, se deben incrustar o vincular al ensamblado emitido por el compilador de Visual J#. El uso de la semántica de .NET Framework para los recursos ofrece tres ventajas principales:

- Entorno totalmente administrado, protegido y con numerosas características para ejecutar aplicaciones
- Compatibilidad incorporada con la localización
- Implementación simplificada de aplicaciones

Cuando se actualizan proyectos existentes de Visual J++ 6.0, se deben convertir primero los recursos que utiliza la aplicación al formato de archivo especificado anteriormente y, a continuación, se deben incrustar o vincular al ensamblado administrado emitido por el compilador de Visual J#. Para obtener más información, vea

[Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#)

Para obtener más información sobre los formatos de archivo de recursos compatibles con .NET Framework, vea

[Recursos en aplicaciones](#).

## Vea también

[Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#) | [Proyectos de Visual J#](#)

# Depurar en Visual J#

Esta sección trata cuestiones específicas de la depuración en Visual J#.

## En esta sección

### [Excepciones de Visual J# con equivalentes de .NET Framework](#)

Enumera las excepciones de Java que tienen equivalentes en .NET Framework, junto con estos equivalentes.

### [Expresiones de Visual J#](#)

Explica cómo utilizar el evaluador de expresiones durante la depuración de código de Visual J#.

## Secciones relacionadas

### [Depurar](#)

Proporciona vínculos a temas sobre la preparación de la depuración, el uso del depurador, depurar código administrado y mucho más.

### [Ejemplos y tutoriales de depuración](#)

Temas paso a paso que incluyen algunas áreas importantes del depurador de Visual Studio. Los tutoriales de depuración son válidos para Visual J#.

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

# Excepciones de Visual J# con equivalentes de .NET Framework

En este tema se explica cómo definir el control de excepciones para el depurador, se proporciona información sobre por qué es necesario definir el control de excepciones y se incluye una tabla de los equivalentes entre Java y .NET Framework.

## Definir control de excepciones en el entorno de desarrollo

El [cuadro de diálogo Excepciones](#), en **Excepciones de Common Language Runtime**, incluye la categoría **Excepciones de Java**. Algunas excepciones de esta categoría tienen equivalentes en .NET Framework, de manera que, si cambia el control de una excepción de Java, hay que cambiarlo también para la excepción equivalente de .NET Framework, con el fin de obtener el mismo comportamiento que con la semántica de excepciones de Java. Esto se debe a que una excepción iniciada en tiempo de ejecución puede provenir de las bibliotecas de clases incluidas en Visual J# o de la capa subyacente de Common Language Runtime de .NET Framework.

## Por qué es necesario definir el control de excepciones

Para el código fuente con código que controla las excepciones, el compilador de Visual J# emite las instrucciones necesarias para asignar las excepciones de .NET Framework a las correspondientes excepciones de la biblioteca de clases, si existen. El código del controlador de excepciones se ejecuta tanto para la excepción de Visual J# como para la correspondiente excepción de .NET Framework. Sin embargo, el depurador no tiene constancia de esta asignación. Por tanto, si se desea interrumpir en una excepción no detectada o cuando se produce una excepción, hay que cambiar el control para la biblioteca de clases y para las correspondientes excepciones de .NET Framework.

## Ejemplo

Para entenderlo mejor, mire el siguiente código de Visual J#:

```
public void caller()
{
    // handler for Visual J# exception
    try {
        stringize(null); // will result in an exception at runtime
    } catch (java.lang.NullPointerException nullEx) {
        // compiler emits code
        // to catch System.NullReferenceException also
        nullEx.printStackTrace();
    }
}

public String stringize(Object o) {
    // invokes method without checking for null
    // results in a System.NullReferenceException at runtime
    return o.toString();
}
```

Si se llama al método **stringize** en tiempo de ejecución con un parámetro null, dará como resultado **System.NullReferenceException**. Sin embargo, el compilador emite el código para controlar tanto a **java.lang.NullPointerException** como a **System.NullReferenceException**. Por tanto, el código de usuario puede detectar la excepción de un modo transparente.

Al depurar con Visual Studio .NET, si se desea cambiar el control para una excepción o un valor null pasado al método **stringize**, no es suficiente con cambiarlo sólo por **java.lang.NullPointerException**. El depurador encontrará **System.NullReferenceException** en tiempo de ejecución; por tanto, se debe cambiar el control para esta excepción también.

Hay que cambiar el control para ambas excepciones, puesto que, si el método **stringize** está en otra clase, se podría iniciar también, explícitamente, **java.lang.NullPointerException**. Para controlar ambas posibilidades, se debe definir el control de ambas excepciones juntas.

## Tabla de excepciones equivalentes

En la siguiente tabla se enumeran las excepciones de Java que tienen equivalentes en .NET Framework, junto con dichos equivalentes.

Excepción de Java	Excepción de .NET Framework
<code>java.lang.ArithmeticException</code>	<code>System.ArithmeticException</code> , <code>System.DivideByZeroException</code> , <code>System.OverflowException</code>
<code>java.lang.ArrayIndexOutOfBoundsException</code>	<code>System.IndexOutOfRangeException</code>
<code>java.lang.ArrayStoreException</code>	<code>System.ArrayTypeMismatchException</code>
<code>java.lang.ClassCastException</code>	<code>System.InvalidCastException</code>
<code>java.lang.NoSuchFieldError</code>	<code>System.MissingFieldException</code>
<code>java.lang.NoSuchMethodError</code>	<code>System.MissingMethodException</code> , <code>System.Reflection.AmbiguousMatchException</code>
<code>java.lang.IncompatibleClassChangeError</code>	<code>System.MissingMemberException</code>
<code>java.lang.IllegalAccessError</code>	<code>System.AccessException</code> , <code>System.FieldAccessException</code> , <code>System.MethodAccessException</code>
<code>java.lang.IllegalArgumentException</code>	<code>System.ArgumentException</code> , <code>System.ArgumentNullException</code> , <code>System.ArgumentOutOfRangeException</code> , <code>System.RankException</code>
<code>java.lang.IllegalMonitorStateException</code>	<code>System.Threading.SynchronizationLockException</code>
<code>java.lang.IllegalThreadStateException</code>	<code>System.Threading.ThreadStateException</code>
<code>java.lang.InternalError</code>	<code>System.NotSupportedException</code>
<code>java.lang.InterruptedIOException</code>	<code>System.Threading.ThreadInterruptedException</code>
<code>java.lang.UnsatisfiedLinkError</code>	<code>System.EntryPointNotFoundException</code>
<code>java.lang.NoClassDefFoundError</code>	<code>System.TypeLoadException</code>
<code>java.lang.NullPointerException</code>	<code>System.NullReferenceException</code>
<code>java.lang.NumberFormatException</code>	<code>System.FormatException</code>
<code>java.lang.OutOfMemoryError</code>	<code>System.OutOfMemoryException</code>
<code>java.lang.SecurityException</code>	<code>System.Security.SecurityException</code>
<code>java.lang.StackOverflowError</code>	<code>System.StackOverflowException</code>
<code>java.lang.VerifyError</code>	<code>System.Security.VerifierException</code>
<code>java.lang.VirtualMachineError</code>	<code>System.ExecutionEngineException</code>

**Vea también**

[Depurar en Visual J#](#) | [Jerarquías de excepciones de Visual J#](#)



# Expresiones de Visual J#

El depurador de Visual J# incluye un evaluador de expresiones que evalúa las expresiones que se escriben en el cuadro de diálogo Inspección rápida de la ventana Inspección, o en las ventanas Automático, Local o This/Me. El evaluador de expresiones también funciona en la ventana Puntos de interrupción. Por ejemplo, si se escribe el nombre de una función donde se desea establecer un punto de interrupción, el evaluador de expresiones la controla.

Los temas siguientes ofrecen información específica y tratan algunos de los tipos de expresión no compatibles:

- [Identificadores y tipos](#)
- [Evaluación de funciones](#)
- [Operadores](#)
- [Matrices](#)
- [Cadenas](#)
- [Conversiones de tipo](#)
- [Comparación y asignación de objetos](#)
- [Evaluación de propiedades](#)
- [WebMethods](#)

El depurador utiliza reglas de ampliación automática para mostrar el contenido de un tipo de datos de un modo coherente. Si es preciso, se pueden agregar elementos de ampliación automática personalizados para mostrar los propios tipos de datos. Para obtener más información, vea [Mostrar elementos de un tipo de datos personalizado](#).

## Identificadores y tipos

Las expresiones del depurador pueden utilizar cualquier identificador visible en el ámbito actual. Si se detiene el depurador en la función `Test`, por ejemplo, se puede utilizar cualquier identificador visible en `Test`, incluidos las constantes, los nombres de variable y los nombres de método.

El depurador puede mostrar correctamente cualquier variable de tipo primitivo o intrínseco. Para variables de tipo `class`, el depurador muestra correctamente el valor según el tipo más derivado. Si se tiene un objeto `leo` de tipo `lion`, derivado del tipo `cat`, se puede evaluar `leo.clawlength` y obtener el valor correcto del objeto de tipo `lion`.

En Visual J#, se puede asignar un valor nuevo a cualquier expresión del lado izquierdo que sea un valor `l` y no una matriz. Esto incluye los tipos **primitive**, **class** y **Object**.

## Evaluación de funciones

El depurador admite la evaluación de funciones, incluidas funciones sobrecargadas. Por tanto, se puede escribir cualquiera de las siguientes expresiones y el depurador llamará a la versión correcta de la función sobrecargada:

```
kangaroo ()
kangaroo (joey)
```

Cuando se evalúa una función en el depurador, en realidad se llama y se ejecuta el código de esa función. Si la función tiene efectos secundarios, como la asignación de memoria o el cambio de valor de una variable global, evaluar la función en la ventana del depurador cambiará el estado del programa, lo que puede producir resultados inesperados.

Cuando se establece un punto de interrupción en una función sobrecargada, la ubicación del punto de interrupción depende de cómo se especifica la función. Si se especifica únicamente el nombre de la función, el depurador establecerá un punto de interrupción en cada sobrecarga de ese nombre de función. Si se especifica la firma completa (nombre de función y lista de argumentos completa), el depurador establecerá un punto de interrupción en la sobrecarga especificada.

No se admite la evaluación de funciones de los siguientes métodos de **java.lang.Object**:

- **wait**
- **notify**
- **notifyAll**
- **getClass**

## Operadores

El depurador evalúa correctamente la mayoría de los operadores integrados, incluidos:

- Operadores relacionales. Ejemplos: ( *expr1* > *expr2*, *expr1* < *expr2*, *expr1* <= *expr2*, *expr1* >= *expr2*, *expr1* == *expr2*, *expr1* != *expr2* ).
- Operadores booleanos. Ejemplos: ( *expr1* && *expr2*, *expr1* || *expr2* )
- Operador condicional. Ejemplo: ( *expr1* ? *expr2* : *expr3* ).
- Operadores aritméticos. Ejemplos: ( *expr1* + *expr2*, *expr1* - *expr2*, *expr1* \* *expr2*, *expr1* / *expr2*, *expr1* % *expr2* ).
- Operadores bit a bit. Ejemplos: ( *expr1* & *expr2*, *expr1* ^ *expr2*, *expr1* | *expr2* ).
- Operadores de desplazamiento. Ejemplos: ( *expr1* >> *expr2*, *expr1* << *expr2*, *expr1* >>> *expr2* ).
- Operadores de asignación. Ejemplos: ( *lvalue* = *expr2*, *lvalue* \* = *expr2*, *lvalue* /= *expr2*, *lvalue* %= *expr2*, *lvalue* += *expr2*, *lvalue* -= *expr2*, *lvalue* <= <= *expr2*, *lvalue* >= >= *expr2*, *lvalue* &= *expr2*, *lvalue* ^= *expr2*, *lvalue* |= *expr2*, *lvalue* >>> = *expr2* ).
- Operadores unarios. Ejemplos: ( +*expr1*, - *expr1*, *expr1*++, ++*expr1*, *expr1*--, --*expr1*, ~*expr1* ).

Puede usar el operador coma para escribir una serie de expresiones: *expr1*, *expr2* y *expr3*.

## Matrices

Visual J# tiene dos formas de matrices: matrices con rango y matrices escalonadas.

Una matriz con rango utiliza un operador [] diferente para cada dimensión:

```
x[ expr1 ][ expr2 ]
```

Una matriz escalonada utiliza sólo un operador [] y comas para separar las dimensiones:

```
x[ expr1, expr2 ]
```

## Cadenas

Las cadenas escritas entre comillas en las ventanas del depurador son equivalentes al tipo **System.String**. El depurador reconoce el operador indizado cuando se utiliza con cadenas y con matrices. Por ejemplo, se puede escribir:

```
"hello world"[0]
```

La ventana Inspección mostrará el valor correcto:

```
'h'
```

En Visual J#, a diferencia de C o C++ nativos, se puede editar el valor de una cadena en el depurador. Además, se puede utilizar el operador `Length` en una cadena:

```
mystring.Length
```

O bien

```
"hello world".Length
```

En Visual J#, se pueden concatenar objetos **String**:

```
"hello" + " world"
```

En Visual J#, se pueden evaluar métodos en cadenas: Por ejemplo, si el código fuente contiene `String str = "Hello World";`, puede evaluar el método **equals** en la ventana Inspección escribiendo la siguiente expresión:

```
str.equals( "Hello World")
```

Esta expresión devuelve true en este ejemplo.

## Conversiones de tipo

En el depurador, funcionan expresiones de conversión de tipos sencillas:

```
(A)x
```

Las conversiones de tipo definidas por el usuario sí funcionan en el depurador.

## Comparación y asignación de objetos

Las expresiones que comparan o asignan objetos de Visual J# funcionan en el depurador:

```
obj1 == obj2  
obj1 = obj2
```

## Evaluación de propiedades

El depurador puede evaluar propiedades en cualquier ventana de variable. Sin embargo, evaluar una propiedad en el depurador puede tener efectos secundarios que produzcan resultados inesperados y no deseados. Para proteger de los efectos secundarios causados por una evaluación accidental, se puede desactivar la evaluación de propiedades en el cuadro de diálogo Opciones.

Las propiedades se pueden evaluar también por medio de sus métodos **get\_** y **set\_**. Por ejemplo, si un objeto tiene una propiedad `text`, `obj.get_text()` mostrará el valor de la propiedad en la ventana Inspección.

## WebMethods

No se puede llamar a WebMethods desde las ventanas del depurador.

## Vea también

[Depurar en Visual J#](#) | [Expresiones en el depurador](#) | [Inspección rápida \(Cuadro de diálogo\)](#) | [Automático \(Ventana\)](#) | [Variables locales \(Ventana\)](#) | [Puntos de interrupción \(Ventana\)](#)

# Implementar aplicaciones de Visual J#

Las aplicaciones creadas con Visual J# se basan en el compilador de Visual J# y Visual J# CodeDOM para la generación y compilación de código dinámico en el servidor. Debe disponer de componentes redistribuibles de Visual J#, es decir, bibliotecas de clases de Visual J#, el compilador de Visual J# y Visual J# CodeDOM, instalados en el servidor donde implementará el servicio Web XML o la aplicación de formularios Web Forms. Utilice Visual J# .NET Redistributable Package para distribuir componentes redistribuibles de Visual J#. Otros aspectos de la implementación son los mismos que para cualquier otra aplicación ASP.NET.

Para obtener más información sobre implementación, vea [Tutoriales sobre implementación](#).

**Nota** Todos los tutoriales sobre implementación se aplican a Visual J#. Sin embargo, los componentes de Visual J# requieren componentes redistribuibles adicionales que se proporcionan con Visual J# .NET Redistributable Package.

## En esta sección

### [Condición de inicio de Visual J#](#)

Describe la forma de asegurarse de que los componentes redistribuibles de Visual J# .NET se han instalado en un equipo de destino.

### [Distribuir Visual J# .NET Redistributable Package](#)

Explica el contenido de Visual J# .NET Redistributable Package.

## Secciones relacionadas

### [Implementar aplicaciones y componentes](#)

Información sobre la creación de archivos ejecutables de instalación, archivos de paquete y la creación de sitios Web.

# Condición de inicio de Visual J#

La condición de inicio de Visual J# .NET se usa para comprobar, durante la instalación, si el paquete Visual J# .NET Redistributable Package se encuentra en el equipo de destino. Esta condición de inicio se agrega automáticamente a un proyecto de implementación cuando se detecta una dependencia de Visual J# .NET Redistributable Package. No se puede quitar.

La condición de inicio de Visual J# .NET busca la versión del paquete Visual J# .NET Redistributable Package especificada en la propiedad **SupportedRuntimes** de la condición de inicio de .NET Framework. Si no se encuentra la condición de inicio de .NET Framework, se usa el valor de **SupportedRuntimes** como versión actual de Visual J# .NET Redistributable Package.

Si no se encuentra la versión requerida de Visual J# .NET Redistributable Package, se detiene la instalación. Se muestra al usuario un cuadro de diálogo **Sí/No** con el texto especificado en la propiedad **Message**. Si el usuario elige **Sí**, se le redirige a la ubicación especificada en la propiedad **InstallUrl**, que es de forma predeterminada un sitio Web de soporte de Microsoft desde el que se puede descargar una copia de Visual J# .NET Redistributable Package. En muchos casos le interesará modificar el valor de la propiedad **InstallUrl** para que apunte a otra ubicación que disponga del archivo redistribuible. Por ejemplo, si distribuye una aplicación en CD-ROM, debería incluir el archivo redistribuible en el CD y cambiar el valor de la propiedad **InstallUrl** para que tenga como valor la ruta de acceso al archivo. Si cambia el valor de la propiedad **InstallUrl**, debería cambiar además la propiedad **Message** para explicar qué se instala y desde dónde.

Puede que tenga que agregar manualmente la condición de inicio de Visual J# .NET a su proyecto de instalación, incluso si no presenta dependencias explícitas con el paquete Visual J# .NET Redistributable Package. Por ejemplo, si los componentes de terceros de Visual J# están comprimidos y disponibles en un programa de instalación, el proyecto de implementación no detectará la dependencia de Visual J#.

## Para agregar a su proyecto de instalación la condición de inicio de Visual J# .NET

1. En el Explorador de soluciones, haga clic con el botón secundario en el proyecto de instalación al que desee agregar la condición de inicio de Visual J# .NET.
2. Haga clic en **Agregar** y después elija **Módulo de combinación**.

Aparece un cuadro de diálogo **Examinar**.

3. Seleccione el archivo VJSRedist\_x86.msm y haga clic en **Abrir**.

Se agregan al proyecto de instalación el módulo de combinación de Visual J# .NET y la condición de inicio.

## Vea también

[Administración de las condiciones de inicio en la implementación](#) | [SupportedRuntimes \(Propiedad\)](#) | [Message \(Propiedad\)](#) | [InstallUrl \(Propiedad\)](#) | [Condición de inicio de .NET Framework](#)

# Distribuir Visual J# .NET Redistributable Package

Las aplicaciones y controles escritos en Visual J# requieren que Visual J# .NET Redistributable Package esté instalado en el equipo donde se ejecuta la aplicación o el control. Para esta versión, los desarrolladores deben redistribuir este paquete junto con sus aplicaciones.

El paquete Visual J# .NET Redistributable Package (vjredist.exe) está disponible en los discos de Visual J# y también se puede descargar desde la dirección <http://msdn.microsoft.com/vjsharp>. Instalar el paquete redistribuible para .NET Framework es un requisito previo para instalar Visual J# .NET Redistributable Package. Para obtener más información sobre el paquete redistribuible para .NET Framework, vea [Distribuir Common Language Runtime](#).

El paquete Visual J# .NET Redistributable Package es compatible con todas las plataformas admitidas por el paquete redistribuible para .NET Framework. No tiene más requisitos del sistema que los que requiere el paquete para .NET Framework.

El programa de instalación de Visual J# .NET Redistributable Package instala los mismos archivos en todas las plataformas compatibles para admitir escenarios tanto de cliente como de servidor.

## Vea también

[Implementar aplicaciones de Visual J#](#) | [Escenarios de implementación](#) | [Implementar aplicaciones](#) | [Distribuir Common Language Runtime](#)

# Referencia

En esta sección, se ofrecen vínculos a información de referencia sobre distintos aspectos de programación en Visual J#.

## En esta sección

### [Opciones del compilador de Visual J#](#)

Hace referencia a las opciones del compilador de Visual J#. Incluye información sobre la generación desde la línea de comandos.

### [Compatibilidad con el lenguaje](#)

Proporciona información de referencia sobre las funciones de Visual J++<sup>®</sup> 6.0 y .NET Framework que son compatibles y no compatibles con Visual J#.

### [Compatibilidad con bibliotecas de clases](#)

Enumera los paquetes compatibles y no compatibles, así como información específica de implementación para la compatibilidad con bibliotecas de clases.

### [Semántica de seguridad para aplicaciones escritas en Visual J#](#)

Trata consideraciones de seguridad para las aplicaciones de Visual J#.

### [Referencia para la actualización a Visual J#](#)

Proporciona información de referencia sobre problemas que pueden surgir al actualizar proyectos de Visual J++ 6.0 a Visual J#.

### [Herramientas de Visual J#](#)

Proporciona información de referencia sobre las herramientas incluidas en Visual J#, como el Conversor binario (Jblmp.exe).

### [Biblioteca de clases de .NET Framework \(sintaxis de Visual J#\)](#)

Proporciona sintaxis de Visual J# para los miembros de la biblioteca de clases de .NET Framework. Cada tema tiene un vínculo con el tema de .NET Framework correspondiente, donde hay información sobre parámetros, valores devueltos y uso.

### [Referencia de la interfaz de usuario](#)

Proporciona información de referencia sobre las características del entorno de desarrollo de Visual Studio<sup>®</sup> que son exclusivas de Visual J#.

## Secciones relacionadas

### [Visual J#](#)

Proporciona vínculos a diversas áreas de la documentación de Visual J#.

### [Buscar información de referencia sobre Java y JDK](#)

Enumera fuentes de material de referencia sobre las bibliotecas de clases de Java y JDK nivel 1.1.4.

### [Buscar material de referencia en la documentación de Visual Studio .NET](#)

Enumera vínculos útiles a temas de la documentación de Visual Studio, Visual C#, Visual Basic y .NET Framework.

# Opciones del compilador de Visual J#

El compilador genera archivos ejecutables (.exe), bibliotecas de vínculos dinámicos (.dll) y archivos de módulos (.netmodule).

Todas las opciones del compilador están disponibles de dos formas: *-opción* y */opción*. En la documentación sólo se muestra el formato */opción*.

## En esta sección

### [Generar desde la línea de comandos](#)

Información sobre cómo generar una aplicación de Visual J# desde la línea de comandos.

### [Opciones del compilador de Visual J# por categoría](#)

Listado por categorías de las opciones del compilador.

### [Opciones del compilador de Visual J# por orden alfabético](#)

Listado alfabético de las opciones del compilador.

### [Opciones del compilador de Visual J++ asignadas a opciones del compilador de Visual J#](#)

Información sobre la asignación de las opciones del compilador de Visual J++® 6.0 a las opciones del compilador de Visual J#.

### [Errores del compilador: de VJS0001 a VJS9999](#)

Documenta los errores generados por el compilador de Visual J#.

## Secciones relacionadas

### [Establecer las propiedades de un proyecto de Visual J#](#)

Enumera los pasos generales para establecer propiedades que controlan cómo se compila, genera y depura un proyecto.

Incluye información sobre los pasos de una generación personalizada.

### [Generaciones predeterminadas y personalizadas](#)

Incluye información sobre los distintos tipos de generación y su configuración.

### [Preparar y administrar generaciones](#)

Contiene procedimientos para generar archivos en el entorno de desarrollo de Visual Studio® .NET.



# Generar desde la línea de comandos

El compilador de Visual J# se puede invocar desde la línea de comandos escribiendo el nombre de su archivo ejecutable (vjc.exe). Es posible que haya que ajustar la ruta de acceso si se desea invocar vjc.exe desde algún subdirectorio del equipo.

Las opciones booleanas, como **/debug**, se pueden habilitar o deshabilitar utilizando + o – en cada opción. Por ejemplo, para que se genere información de depuración, se puede especificar **/debug+** (o sólo **/debug**) y, para deshabilitarla, se puede especificar **/debug-**.

En la línea de comandos, se pueden mezclar opciones con los archivos de código fuente.

Este tema proporciona detalles sobre los siguientes apartados:

- [Reglas para la sintaxis de la línea de comandos](#)
- [Líneas de comandos de ejemplo](#)

## Reglas para la sintaxis de la línea de comandos

El compilador de Visual J# utiliza las siguientes reglas al interpretar los argumentos empleados en la línea de comandos del sistema operativo:

- Los argumentos van delimitados por espacio en blanco, que puede ser un carácter de espacio o una tabulación.
- El carácter de intercalación (^) no se reconoce como carácter de escape ni como delimitador. El analizador de la línea de comandos del sistema operativo procesa este carácter por completo antes de pasarlo a la matriz argv del programa.
- Una cadena entre comillas ("cadena") se interpreta como un solo argumento, sin importar el espacio en blanco que contenga. Se puede incrustar una cadena entre comillas dentro de un argumento.
- Las comillas dobles precedidas por una barra diagonal inversa (\") se interpretan como un carácter literal de comillas dobles (").
- Las barras diagonales inversas se interpretan literalmente, a menos que precedan a unas comillas dobles.
- Si a un grupo de barras invertidas (en número par) le siguen comillas dobles, se colocará una barra diagonal inversa en la matriz argv por cada par de barras diagonales inversas, y las comillas dobles se interpretarán como un delimitador de cadenas.
- Si a un grupo de barras diagonales inversas (en número impar) le siguen comillas dobles, se colocará una barra diagonal inversa en la matriz argv por cada par de barras inversas, y la última barra diagonal inversa se interpretará como un carácter de escape de las comillas dobles, por lo que se colocará un literal de comillas dobles (") en argv.

## Líneas de comandos de ejemplo

- Para compilar File.jsl y crear File.exe, ejecute:

```
vjc File.jsl
```

- Para compilar File.jsl y crear File.dll, ejecute:

```
vjc /target:library File.jsl
```

- Para compilar File.jsl y producir My.exe, ejecute:

```
vjc /out:My.exe File.jsl
```

- Para compilar todos los archivos \*.jsl del directorio actual y definir el símbolo DEBUG. El resultado es File2.exe:

```
vjc /define:DEBUG /out:File2.exe *.jsl
```

- Para compilar todos los archivos de Visual J# en el directorio actual y producir una versión de depuración de File2.dll (se utiliza la forma abreviada de la opción **/target**). No se muestra ningún logotipo:

```
vjc /t:1 /out:File2.dll /nologo /debug *.jsl
```

- Para compilar todos los archivos \*.jsl del directorio actual y de todos los subdirectorios, y dar lugar a MyLibrary.dll,

ejecute:

```
vjc /target:library /recurse:*.jsl /out:MyLibrary.dll
```

- Para compilar `main.jsl` e importar las clases de `MyLibrary.dll`, ejecute:

```
vjc /reference:MyLibrary.dll main.jsl
```

### **Vea también**

[Opciones del compilador de Visual J#](#) | [Opciones del compilador de Visual J# por orden alfabético](#) | [Opciones del compilador de Visual J# por categoría](#)

# Opciones del compilador de Visual J# por categoría

Las siguientes opciones del compilador están ordenadas por categoría. Para obtener una lista por orden alfabético, vea [Opciones del compilador de Visual J# por orden alfabético](#).

## Optimización

Opción	Propósito
<a href="#">/optimize</a>	Habilita o deshabilita las optimizaciones.

## Archivos de resultados

Opción	Propósito
<a href="#">/out</a>	Especifica el archivo de resultados.
<a href="#">/target</a>	Especifica el formato del archivo de resultados utilizando una de estas opciones: <a href="#">/target:exe</a> <a href="#">/target:library</a> <a href="#">/target:module</a> <a href="#">/target:winexe</a>

## Ensamblados de .NET Framework

Opción	Propósito
<a href="#">/libpath</a>	Especifica la ubicación de los ensamblados a los que se hace referencia mediante <a href="#">/reference</a> .
<a href="#">/reference</a>	Importa metadatos de un archivo que contiene un ensamblado.
<a href="#">/securescoping</a>	Crea miembros con ámbito de paquete internos al ensamblado.

## Depuración y comprobación de errores

Opción	Propósito
<a href="#">/debug</a>	Proporciona información de depuración.
<a href="#">/nowarn</a>	Anula la capacidad del compilador de generar determinadas advertencias.
<a href="#">/warn</a>	Establece el nivel de advertencia.
<a href="#">/warnaserror</a>	Convierte las advertencias en errores.

## Preprocesador

Opción	Propósito
<a href="#">/define</a>	Define los símbolos de preprocesador.

## Recursos

Opción	Propósito
<a href="#">/linkresource</a>	Crea un vínculo con un recurso de .NET Framework en el archivo de resultados.
<a href="#">/resource</a>	Inserta un recurso de .NET Framework en el archivo de resultados.
<a href="#">/win32res</a>	Inserta un recurso Win32 en el archivo de resultados.

## Varios

Opción	Propósito
<a href="#">@</a>	Especifica un archivo de respuesta.
<a href="#">/?</a>	Muestra las opciones del compilador en stdout.
<a href="#">/baseaddress</a>	Especifica la dirección base preferente en la que se debe cargar un archivo DLL.
<a href="#">/codepage</a>	Especifica la página de códigos que se debe utilizar para todos los archivos de código fuente de la compilación.
<a href="#">/help</a>	Muestra las opciones del compilador en stdout.
<a href="#">/jcpa</a>	Proporciona asociaciones de paquete de Java y COM.

<a href="#">/main</a>	Especifica la ubicación del método main.
<a href="#">/nologo</a>	Suprime la pantalla de bienvenida del compilador.
<a href="#">/recurse</a>	Busca en los subdirectorios archivos de código fuente para compilar.
<a href="#">/utf8output</a>	Muestra los resultados del compilador mediante la codificación UTF-8.
<a href="#">/x</a>	Deshabilita las extensiones de lenguaje

**Vea también**

[Opciones del compilador de Visual J#](#) | [Opciones del compilador de Visual J# por orden alfabético](#) | [Generar desde la línea de comandos](#)

# Opciones del compilador de Visual J# por orden alfabético

Las siguientes opciones del compilador están ordenadas alfabéticamente. Para obtener una lista por categoría, vea [Opciones del compilador de Visual J# por categoría](#).

Opción	Propósito
@	Especifica un archivo de respuesta.
/?	Muestra las opciones del compilador en stdout.
/baseaddress	Especifica la dirección base preferente en la que se debe cargar un archivo DLL.
/codepage	Especifica la página de códigos que se debe utilizar para todos los archivos de código fuente de la compilación.
/debug	Proporciona información de depuración.
/define	Define los símbolos de preprocesador.
/help	Muestra las opciones del compilador en stdout.
/jcpa	Proporciona asociaciones de paquete de Java y COM.
/libpath	Especifica la ubicación de los ensamblados a los que se hace referencia mediante <a href="#">/reference</a> .
/linkresource	Crea un vínculo con un recurso de .NET Framework en el archivo de resultados.
/main	Especifica la ubicación del método main.
/nologo	Suprime la pantalla de bienvenida del compilador.
/nowarn	Anula la capacidad del compilador de generar determinadas advertencias.
/optimize	Habilita o deshabilita las optimizaciones.
/out	Especifica el archivo de resultados.
/recurse	Busca en los subdirectorios archivos de código fuente para compilar.
/reference	Importa metadatos de un archivo que contiene un ensamblado.
/resource	Inserta un recurso de .NET Framework en el archivo de resultados.
/securescoping	Crea miembros con ámbito de paquete internos al ensamblado.
/target	Especifica el formato del archivo de resultados utilizando una de estas opciones: <a href="#">/target:exe</a> <a href="#">/target:library</a> <a href="#">/target:module</a> <a href="#">/target:winexe</a>
/utf8output	Muestra los resultados del compilador mediante la codificación UTF-8.
/warn	Establece el nivel de advertencia.
/warnaserror	Convierte las advertencias en errores.
/win32res	Inserta un recurso Win32 en el archivo de resultados.
/x	Deshabilita las extensiones de lenguaje

## Vea también

[Opciones del compilador de Visual J#](#) | [Opciones del compilador de Visual J# por categoría](#) | [Generar desde la línea de comandos](#)

# @ (Especificar archivo de respuesta)

```
@response_file
```

donde:

*response\_file*

Archivo que especifica opciones del compilador o archivos de código fuente para compilar.

## Comentarios

La opción @ permite especificar un archivo que contiene opciones del compilador y archivos de código fuente para compilar. Estas opciones de compilador y archivos de código fuente serán procesados por el compilador como si fuesen especificados en la línea de comandos.

Para especificar varios archivos de respuesta en una compilación, hay que especificar varias opciones de archivo de respuesta. Por ejemplo:

```
@file1.rsp @file2.rsp
```

En una misma línea de un archivo de respuesta, pueden aparecer varias opciones del compilador y archivos de código fuente. Una especificación de opción del compilador debe aparecer en una única línea (no puede abarcar varias líneas). Los archivos de respuesta pueden contener comentarios que empiezan con el símbolo #.

Especificar opciones del compilador desde un archivo de respuesta produce el mismo efecto que incluir esas opciones en la línea de comandos. Para obtener más información, vea [Generar desde la línea de comandos](#).

El compilador procesa las opciones de comandos según las encuentra. Por consiguiente, los argumentos de la línea de comandos pueden reemplazar opciones enumeradas anteriormente en archivos de respuesta. A la inversa, las opciones de un archivo de respuesta reemplazarán opciones incluidas previamente en la línea de comandos o en otros archivos de respuesta.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

A continuación, se muestran algunas líneas de un archivo de respuesta de ejemplo:

```
# build the first output file
/target:exe /out:MyExe.exe source1.jsl source2.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /baseaddress (Especificar la dirección base de un archivo DLL)

`/baseaddress:address`

donde:

*address*

Dirección base para el archivo DLL. Esta dirección se debe especificar en forma de número hexadecimal.

## Comentarios

La opción **/baseaddress** permite especificar la dirección base preferida para cargar una DLL. La dirección base predeterminada para un archivo DLL la establece el motor Common Language Runtime de .NET Framework.

Tenga en cuenta que se redondeará la palabra de orden inferior de esta dirección. Por ejemplo, si se especifica 0x11110001, se redondeará a 0x11110000.

Esta opción no se tiene en cuenta si el destino no es un archivo DLL.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Dirección base**.

## Para establecer esta opción del compilador mediante programación

Vea [BaseAddress \(Propiedad\)](#).

## Vea también

[Opciones del compilador de Visual J#](#)

# /codepage (Especificar una página de códigos para los archivos de código fuente)

`/codepage:id`

donde:

*id*

El identificador de la página de códigos para todos los archivos de código fuente en la compilación.

## Comentarios

Si se compilan uno o varios archivos de código fuente que no se crearon para utilizar la página de códigos predeterminada en el equipo, se podrá utilizar la opción **/codepage** para especificar la página de códigos que debe utilizarse. **/codepage** se aplica a todos los archivos de código fuente de la compilación.

Si los archivos de código fuente se crearon con la misma página de códigos que está activada en el equipo o se crearon con UNICODE o UTF-8, no es preciso utilizar **/codepage**.

Para obtener más información sobre cómo averiguar qué páginas de códigos admite el sistema, vea [GetCPInfo](#).

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Vea también

[Opciones del compilador de Visual J#](#)



# /debug (Emitir información de depuración)

```
/debug[+ | -]  
/debug:{full | pdbonly}
```

donde:

**+ | -**  
Si se especifica **+**, o simplemente **/debug**, el compilador genera información de depuración y la incluye en una base de datos de programa (archivo .pdb). Si se especifica **-**, que es la opción predeterminada si no se especifica **/debug**, no se crea información de depuración.

**full | pdbonly**

Especifica el tipo de información de depuración generada por el compilador. El argumento *full*, que es la opción predeterminada si no se especifica **/debug:pdbonly**, permite asociar un depurador al programa que se ejecuta. Al especificar la opción *pdbonly*, se permite depurar el código fuente cuando se inicia el programa en el depurador, pero sólo muestra el ensamblador cuando el programa que se ejecuta está asociado al depurador.

## Comentarios

La opción **/debug** da lugar a que el compilador genere información de depuración y la incluya en los archivos resultantes. Utilice esta opción para generar versiones de depuración. Si no se especifica **/debug**, **/debug+** ni **/debug:full**, no se podrá depurar el archivo de resultados del programa.

Para obtener información sobre cómo configurar el rendimiento de depuración de una aplicación, vea [Facilitar la depuración de una imagen](#).

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique la propiedad **Generar información de depuración**.

## Para establecer esta opción del compilador mediante programación

Vea [DebugSymbols \(Propiedad\)](#).

## Ejemplo

Para incluir información de depuración en el archivo de resultados `app.exe`, ejecute:

```
vjc /debug /out:app.exe test.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /define (Definición de preprocesador)

```
/define:name[=value][;name2[=value]]
```

donde:

*name*, *name2*

Nombre de uno o varios símbolos que desea definir.

*value*

Valor opcional que se asigna al símbolo.

## Comentarios

La opción **/define** define *name* como un símbolo en el programa. Tiene el mismo efecto que el uso de la directiva de preprocesador **#define** en el archivo de código fuente. Un símbolo permanece definido hasta que una directiva **#undef** del archivo de código fuente quita la definición o hasta que el compilador llega al final del archivo.

Se pueden utilizar símbolos creados por esta opción con **#if**, **#else**, **#elif** y **#endif** para compilar archivos de código fuente de forma condicional.

**/d** es la forma abreviada de **/define**.

Se pueden definir múltiples símbolos con **/define** utilizando punto y coma o coma para separar los nombres de símbolos. Por ejemplo:

```
/define:DEBUG;TUESDAY
```

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique el valor de la propiedad **Constantes de compilación condicional**.

## Para establecer esta opción del compilador mediante programación

Vea [DefineConstants \(Propiedad\)](#).

## Ejemplo

```
// vjc_compiler_define.jsl
// compile with: /define:DEBUG
// or uncomment the next line
// #define DEBUG
public class Test
{
    public static void main(String args[])
    {
        #if (DEBUG)
            System.Console.WriteLine("DEBUG defined");
        #else
            System.Console.WriteLine("DEBUG not defined");
        #endif
    }
}
```

## Salida

```
DEBUG defined
```

**Vea también**

[Opciones del compilador de Visual J#](#)

# /help, /? (Ayuda de la línea de comandos del compilador)

```
/help  
/?
```

## Comentarios

Esta opción envía una lista de opciones del compilador y una breve descripción de cada opción a stdout. Si se incluye esta opción en una compilación, no se creará ningún archivo de resultados y no se llevará a cabo la compilación.

### Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

Esta opción del compilador no está disponible en Visual Studio.

### Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Vea también

[Opciones del compilador de Visual J#](#)

# /jcpa (Asociar paquetes de Java y COM)

```
/jcpa:package=namespace  
/jcpa:@filename
```

donde:

*package*

Nombre del paquete de una clase o interfaz COM.

*namespace*

Espacio de nombres del contenedor administrado al que se desea asignar el paquete.

*filename*

Archivo que contiene una o más asociaciones *package=namespace*. En el archivo, hay que separar los pares con una línea nueva o con punto y coma.

## Comentarios

La opción **/jcpa** es necesaria cuando dos o más DLL COM a las que tiene acceso la aplicación contienen coclases o interfaces con el mismo nombre. Esta situación tiene lugar cuando se actualizan aplicaciones de Java o COM de Visual J++ 6.0 a Visual J#.

Cuando se compilan contenedores generados con JactiveX para archivos de código fuente de Java o COM, se puede tener una interfaz COM con el mismo nombre en más de un paquete. En este caso, el tipo administrado correspondiente a esta interfaz COM estará presente en varios ensamblados generados con la herramienta Importador de la biblioteca de tipos (Tlbimp.exe). En esta situación, vjc.exe no podrá asignar estas interfaces COM con los tipos administrados correctos y dará un error en tiempo de compilación; se puede utilizar **/jcpa** para resolver este conflicto.

Vea [Información general: actualizar proyectos de Java o COM](#) para obtener más información.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

Para compilar `in.jsl` y asociar el paquete `x` con el espacio de nombres `y`, ejecute:

```
vjc /jcpa:x=y in.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /libpath (Especificar ubicaciones de referencias a ensamblados)

```
/libpath:dir1[; dir2]
```

donde:

*dir1*

Directorio utilizado por el compilador para buscar un ensamblado al que se hace referencia si no lo encuentra en el directorio de trabajo actual (el directorio desde el que se invoca al compilador) o en el directorio del sistema de Common Language Runtime.

*dir2*

Uno o varios directorios adicionales para buscar las referencias a ensamblados. Separe los nombres de directorios adicionales con punto y coma.

## Comentarios

La opción **/libpath** especifica la ubicación de ensamblados a los que se hace referencia mediante la opción [/reference](#).

**/libpath** señala a la variable de entorno LIB.

El compilador busca referencias a ensamblados que no presentan la ruta completa en el siguiente orden:

1. Directorio actual de trabajo. Es el directorio desde donde se invoca al compilador.
2. Directorio del sistema de Common Language Runtime.
3. Directorio de sistema de Visual J#.
4. Directorios especificados por **/libpath**.
5. Directorios especificados por la variable de entorno LIB.

Hay que utilizar [/reference](#) para especificar una referencia a un ensamblado.

**/libpath** es aditiva; si se especifica más de una vez, se anexa a los valores anteriores.

Una alternativa al uso de **/libpath** consiste en copiar en el directorio de trabajo los ensamblados requeridos; esto permitirá pasar el nombre del ensamblado a **/reference**. A continuación, se pueden eliminar los ensamblados del directorio de trabajo.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **Ruta de acceso de referencias**.
4. Modifique el contenido del cuadro de lista.

## Para establecer esta opción del compilador mediante programación

Vea [ReferencePath \(Propiedad\)](#).

## Ejemplo

Compila `t2.jsl` para crear un archivo `.exe`. El compilador buscará referencias a ensamblados en el directorio de trabajo y en el directorio de ensamblados de la unidad C.

```
vjc /libpath:c:\assemblies /reference:t2.dll t2.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /linkresource (Vincular a recursos de .NET Framework)

```
/linkresource:filename [,identifier]
```

donde:

*filename*

Archivo de recursos de .NET Framework con el que desea crear un vínculo desde el ensamblado.

*identifier* (opcional)

Nombre lógico del recurso; nombre usado para cargar el recurso. El valor predeterminado es el nombre del archivo. Sólo se puede especificar un nombre lógico para un recurso.

## Comentarios

La opción **/linkresource** crea un vínculo con un archivo de recursos de .NET Framework en el archivo de resultados, pero el archivo de recursos no se coloca en el archivo de resultados. [/resource](#), en cambio, incrusta un archivo de recursos en el archivo de resultados.

Los recursos vinculados son públicos en el ensamblado cuando se crean con el compilador de Visual J#.

**/linkresource** requiere una de las opciones [/target](#) que no sea [/target:module](#).

Si *filename* es un archivo de recursos de .NET Framework creado, por ejemplo, con [Resgen.exe](#) o en el entorno de desarrollo, se puede tener acceso a él con miembros del espacio de nombres **System.Resources** (vea [System.Resources.ResourceManager](#) para obtener más información). En todos los demás recursos, hay que utilizar los métodos **GetManifestResource\*** en la clase **System.Reflection.Assembly** para tener acceso al recurso en tiempo de ejecución.

*filename* puede tener cualquier formato de archivo. Por ejemplo, se puede hacer que una DLL nativa forme parte de un ensamblado para que se pueda instalar en la Caché de ensamblados global y sea accesible desde código administrado del ensamblado.

La opción **/linkres** es la forma breve de **/linkresource**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

Para compilar `in.jsl` y vincularlo al archivo de recursos `rf.resource`, ejecute:

```
vjc /linkresource:rf.resource in.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

## /main (Especificar la ubicación del método main)

```
/main:class
```

donde:

*class*

Clase que contiene el método main.

### Comentarios

Si la compilación incluye más de una clase con un método main, se puede especificar la clase que contiene el método main que se desea utilizar como punto de entrada en el programa. El nombre de clase especificado en el argumento puede ser un nombre sencillo o un nombre completo que incluya el nombre del paquete.

Esta opción sólo se puede usar al compilar archivos .exe.

### Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **General**.
4. Modifique el valor de la propiedad **Objeto inicial**.

### Para establecer esta opción del compilador mediante programación

Vea [StartupObject \(Propiedad\)](#).

### Ejemplo

Para compilar `ClassA.jsl` y `ClassB.jsl`, y especificar que el método main está en `ClassA`, ejecute:

```
vjc ClassA.jsl ClassB.jsl /main:ClassA
```

### Vea también

[Opciones del compilador de Visual J#](#)



# /nologo (Suprimir la información de pancarta)

/nologo[+ | -]

donde:

+ | -

Si se especifica **/nologo+** o sólo **/nologo**, se da lugar a que el compilador no muestre la información de pancarta. Si se especifica **/nologo-**, que está activo si no se especifica esta opción del compilador, se da lugar a que sí se muestre la pancarta.

## Comentarios

La opción **/nologo** suprime la presentación de la pancarta de inicio de sesión cuando se ejecuta el compilador y la de los mensajes de información durante la compilación.

Esta opción no está disponible en el entorno de desarrollo; sólo está disponible al compilar desde la línea de comandos.

### Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

Esta opción del compilador no está disponible en Visual Studio.

### Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Vea también

[Opciones del compilador de Visual J#](#)

# /nowarn (Suprimir las advertencias especificadas)

`/nowarn:number1[,number2[...]]`

donde:

*number1*, *number2*

Número de advertencia que se desean que suprima el compilador.

## Comentarios

La opción **/nowarn** permite suprimir la capacidad del compilador para generar una o más advertencias. Si hay varios números de advertencia, hay que separarlos con una coma.

Sólo hay que especificar la parte numérica del identificador de advertencia. Por ejemplo, si se desea suprimir VJS0028, hay que especificar `/nowarn:28`.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique el valor de la propiedad **Suprimir advertencias específicas**.

## Para establecer esta opción del compilador mediante programación

Vea [NoWarn](#).

## Vea también

[Opciones del compilador de Visual J#](#)

# /optimize (Habilitar o deshabilitar optimizaciones)

```
/optimize[+ | -]
```

donde:

+ | -

De forma predeterminada, **/optimize** está activado. Para evitar la creación de un archivo de resultados optimizado, especifique **/optimize-**. Especificar **/optimize** equivale a especificar **/optimize+**.

## Comentarios

La opción **/optimize** habilita o deshabilita las optimizaciones realizadas por el compilador para hacer que el archivo de resultados sea más pequeño, rápido y eficiente. También indica a Common Language Runtime que optimice el código en tiempo de ejecución.

**/o** es la forma breve de **/optimize**.

Se pueden combinar las opciones **/optimize** y [/debug](#).

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique la propiedad **Optimizar código**.

## Para establecer esta opción del compilador mediante programación

Vea [Optimize \(Propiedad\)](#).

## Ejemplo

Para compilar `t2.js1` y habilitar las optimizaciones del compilador, ejecute:

```
vjc t2.js1 /optimize
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /out (Definir el nombre del archivo de resultados)

```
/out:filename
```

donde:

*filename*

Nombre del archivo de resultados creado por el compilador.

## Comentarios

La opción **/out** especifica el nombre del archivo de resultados.

Si no se especifica el nombre del archivo de resultados:

- Un archivo .exe toma el nombre del archivo de código fuente que contiene el método main del punto de entrada.
- Un archivo .dll toma el nombre del primer archivo de código fuente.

Hay que especificar el nombre completo y la extensión del archivo que se desea crear.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **General**.
4. Modifique la propiedad **Nombre del ensamblado**.

## Para establecer esta opción del compilador mediante programación

Vea [OutputFileName \(Propiedad\)](#).

## Ejemplo

Para compilar `t1.jsl` y `t2.jsl`, y crear el archivo de resultados `app.exe`, ejecute:

```
vjc t1.jsl t2.jsl /out:app.exe
```

## Vea también

[Opciones del compilador de Visual J#](#)

## /recurse (Buscar archivos de código fuente en subdirectorios)

```
/recurse:[dir\]file
```

donde:

*dir* (opcional)

Directorio en el que se desea iniciar la búsqueda. Si no se especifica, la búsqueda empieza en el directorio del proyecto. No se permiten caracteres comodín.

*file*

Los archivos que se van a buscar. Se permiten caracteres comodín.

### Comentarios

La opción **/recurse** permite compilar archivos de código fuente de todos los directorios secundarios del directorio especificado (*dir*) o del directorio del proyecto. Sólo se puede especificar una combinación *dir\file* con esta opción. Si se desea especificar otro par *dir\file*, hay que utilizar una opción **/recurse** distinta.

El nombre de archivo se puede especificar con caracteres comodín de modo que se compilen todos los archivos del directorio del proyecto que cumplan la especificación sin tener que utilizar **/recurse**.

### Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

### Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

### Ejemplo

Para compilar todos los archivos de código fuente .jsl del directorio actual, ejecute:

```
vjc *.jsl
```

Para compilar todos los archivos de código fuente .jsl del directorio *dir1\dir2* y de los directorios que contenga, y generar *dir2.dll*, ejecute:

```
vjc /target:library /out:dir2.dll /recurse:dir1\dir2\*.jsl
```

### Vea también

[Opciones del compilador de Visual J#](#)

# /reference (Importar metadatos)

```
/reference:file[;file2]
```

donde:

*file*, *file2*

Uno o varios archivos que contienen un manifiesto de ensamblado. Para importar más de un archivo, hay que separar los nombres de archivo con comas o puntos y comas.

## Comentarios

La opción **/reference** da lugar a que el compilador haga pública la información de tipos de los archivos especificados para el proyecto que se está compilando.

En tiempo de ejecución, debe tener en cuenta que sólo se puede cargar un ensamblado .exe por cada proceso, aunque a veces se podrá cargar más de un .exe en un mismo proceso. Por tanto, se recomienda no pasar un ensamblado generado con **/target:exe** o **/target:winexe** a **/reference** si se compila con las opciones **/target:winexe** o **/target:exe**. Esta situación puede modificarse en versiones futuras de Common Language Runtime.

Si se hace referencia a un ensamblado (Ensamblado A) que, a su vez, hace referencia a otro ensamblado (Ensamblado B), se deberá hacer referencia al ensamblado B si:

- Un tipo del Ensamblado A que se utiliza se hereda de un tipo o implementa una interfaz del Ensamblado B, e intenta utilizar un miembro del tipo o interfaz base del Ensamblado B.
- Si se invoca un campo, una propiedad, un evento o un método que devuelve un tipo o tiene un tipo de parámetro de Ensamblado B.

Se utiliza [/libpath](#) para especificar el directorio en el que se encuentran una o varias de las referencias de ensamblados. El tema [/libpath](#) trata también sobre los directorios donde el compilador busca ensamblados.

Para que el compilador reconozca un tipo en un ensamblado (no en un módulo), debe obligársele a que resuelva el tipo, lo que se puede conseguir, por ejemplo, definiendo una instancia del tipo. Existen otras formas de que el compilador resuelva nombres de tipos en un ensamblado; por ejemplo, si se hereda de un tipo de un ensamblado, el compilador reconocerá el nombre del tipo.

**/r** es la forma abreviada de **/reference**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

Vea [Agregar referencia \(Cuadro de diálogo\)](#).

## Para establecer esta opción del compilador mediante programación

Vea [Add \(Método, objeto References\)](#).

## Ejemplo

Para compilar el archivo de código fuente `input.jsl` e importar metadatos de `metad1.dll` y `metad2.dll` para generar `out.exe`, ejecute:

```
vjc /reference:metad1.dll;metad2.dll /out:out.exe input.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /resource (Incrustar un archivo de recursos en el resultado)

```
/resource:filename[,identifier]
```

donde:

*filename*

Archivo de recursos de .NET Framework que se desean incrustar en el archivo de resultados de la compilación.

*identifier* (opcional)

Nombre lógico del recurso; nombre usado para cargar el recurso. El valor predeterminado es el nombre del archivo. Sólo se puede especificar un nombre lógico para un recurso.

## Comentarios

Se utiliza la opción [/linkresource](#) para vincular un recurso a un ensamblado y no incluir el archivo de recursos en el archivo de resultados.

Los recursos son públicos en el ensamblado cuando se crean con el compilador de Visual J#.

Si *filename* es un archivo de recursos de .NET Framework creado, por ejemplo, con [Resgen.exe](#) o en el entorno de desarrollo, se puede tener acceso a él con miembros del espacio de nombres **System.Resources** (vea [System.Resources.ResourceManager](#) para obtener más información). En todos los demás recursos, hay que utilizar los métodos **GetManifestResource\*** en la clase **System.Reflection.Assembly** para tener acceso al recurso en tiempo de ejecución.

**/res** es la forma abreviada de **/resource**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Agregue un archivo de recursos al proyecto.
2. Seleccione el archivo que desea incrustar en el Explorador de soluciones.
3. Seleccione **Acción de generación** para ese archivo en la ventana Propiedades.
4. Establezca el valor de **Acción de generación** en **Recurso incrustado**.

## Para establecer esta opción del compilador mediante programación

Vea [BuildAction \(Propiedad\)](#).

## Ejemplo

Para compilar `in.jsl` y asociar el archivo de recursos `rf.resource`, ejecute:

```
vjc /resource:rf.resource in.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /seurescoping (Hacer inaccesibles los miembros con ámbito de paquete fuera del ensamblado)

```
/seurescoping[+ | -]
```

donde:

+ | -

De forma predeterminada, la opción **/seurescoping** está activada. Para marcar como públicos metadatos de elementos con ámbito de paquete, hay que especificar **/seurescoping-**. Especificar **/seurescoping+** equivale a especificar **/seurescoping**.

## Comentarios

La opción **/seurescoping** permite limitar el ámbito de las clases con ámbito de paquete a un paquete.

Las clases se asocian con paquetes. Cuando el ensamblado contiene un paquete al que se hace referencia en otra compilación, los miembros con ámbito de paquete están disponibles para las clases de un paquete con el mismo nombre.

Cuando se especifica la opción **/seurescoping**, el ámbito de paquete se trata como un ámbito de ensamblado cuando se emiten metadatos para un miembro. Hay que tener en cuenta que los miembros con ámbito de paquete incluyen miembros marcados con el modificador protegido.

**/ss** es la forma breve de **/seurescoping**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

Para compilar el archivo de código fuente `input.jsl` y hacer inaccesibles los tipos de paquete cuando se haga referencia al ensamblado, ejecute:

```
vjc /seurescoping /target:library input.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)



# **/target (Especificar el formato del archivo de resultados)**

La opción del compilador **/target** se puede especificar de cuatro formas distintas:

[/target:exe](#)

Crea un archivo .exe de consola.

[/target:library](#)

Crea una biblioteca de códigos.

[/target:module](#)

Crea un módulo.

[/target:winexe](#)

Crea un archivo .exe de Windows.

La opción **/target** hace que se incluya un manifiesto de [ensamblado](#) de .NET Framework en un archivo de resultados. Cada proyecto de Visual J# incluye un archivo denominado AssemblyInfo.jsl. Vea este archivo si desea obtener una plantilla para modificar metadatos de ensamblado.

Cada invocación de vjc.exe produce como máximo un archivo de resultados. Si especifica una opción de compilador como **/out** o **/target** más de una vez, el compilador activará el último que lea. Se coloca información sobre todos los archivos de una compilación en el manifiesto. Todos los archivos de resultados contienen metadatos de ensamblado en el manifiesto. Se utiliza el [Desensamblador de MSIL \(Ildasm.exe\)](#) para ver los metadatos de un archivo de resultados.

## **Para establecer esta opción del compilador mediante programación**

Vea [OutputType \(Propiedad\)](#).

## **Vea también**

[Opciones del compilador de Visual J#](#)

# /target:exe (Crear una aplicación de consola)

```
/target:exe
```

## Comentarios

La opción **/target:exe** hace que el compilador cree una aplicación de consola ejecutable (EXE). La opción **/target:exe** está activada de manera predeterminada. El archivo ejecutable se creará con la extensión .exe.

Se utiliza [/target:winexe](#) para crear un programa ejecutable de Windows.

A menos que se especifique otra cosa con la opción [/out](#), el archivo de resultados toma el nombre del archivo de entrada que contiene el método main.

Sólo se requiere un método main en los archivos de código fuente que se compilan para producir un .exe. La opción [/main](#) del compilador permite especificar la clase que contiene el método main, en casos en los que el código tiene más de una clase con un método main.

**/t:exe** es la forma breve de **/target:exe**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **General**.
4. Modifique la propiedad **Tipo de resultado**.

## Para establecer esta opción del compilador mediante programación

Vea [OutputType \(Propiedad\)](#).

## Ejemplo

Cada una de las siguientes líneas de comandos compilará `in.cs` y creará `in.exe`:

```
vjc /target:exe in.jsl  
vjc in.jsl
```

## Vea también

[/target](#) (Especificar el formato del archivo de resultados) | [Opciones del compilador de Visual J#](#)

# /target:library (Crear una biblioteca de códigos)

```
/target:library
```

## Comentarios

La opción **/target:library** hace que el compilador cree una biblioteca de vínculos dinámicos (DLL) en lugar de un archivo ejecutable (EXE). Se creará el archivo DLL con la extensión .dll.

A menos que se especifique otra cosa con la opción [/out](#), el archivo de resultados toma el nombre del primer archivo de entrada.

Para generar un archivo .dll, no es necesario un método main.

**/t:l** es la forma breve de **/target:library**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **General**.
4. Modifique la propiedad **Tipo de resultado**.

## Para establecer esta opción del compilador mediante programación

Vea [OutputType \(Propiedad\)](#).

## Ejemplo

Para compilar `in.jsl` y crear `in.dll`, ejecute:

```
vjc /target:library in.jsl
```

O bien

```
vjc /t:l in.jsl
```

## Vea también

[/target](#) (Especificar el formato del archivo de resultados) | [Opciones del compilador de Visual J#](#)

# **/target:module (Crear un módulo para agregarlo a un ensamblado)**

```
/target:module
```

## **Comentarios**

Si no se desea generar un manifiesto de ensamblado en un archivo de resultados del Lenguaje intermedio de Microsoft (MSIL), hay que utilizar la opción **/target:module**. De manera predeterminada, el archivo de resultados tendrá la extensión .netmodule.

Common Language Runtime de .NET Framework no puede cargar un archivo sin ensamblado.

## **Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio**

El entorno de desarrollo de Visual Studio no permite la creación de módulos.

## **Para establecer esta opción del compilador mediante programación**

Vea [OutputType \(Propiedad\)](#).

## **Ejemplo**

Para compilar `in.jsl` y crear `in.netmodule`, ejecute:

```
vjc /target:module in.jsl
```

## **Vea también**

[/target \(Especificar el formato del archivo de resultados\)](#) | [Opciones del compilador de Visual J#](#)

# /target:winexe (Crear un programa para Windows)

```
/target:winexe
```

## Comentarios

La opción **/target:winexe** hace que el compilador cree un programa de Windows ejecutable (EXE). El archivo ejecutable se creará con la extensión .exe. Los programas para Windows proporcionan una interfaz de usuario para la biblioteca de .NET Framework o con las API Win32.

Se utiliza [/target:exe](#) para crear una aplicación de consola.

A menos que se especifique otra cosa con la opción [/out](#), el archivo de resultados toma el nombre del archivo de entrada que contiene el método main.

Sólo se requiere un método main en los archivos de código fuente que se compilan para producir un .exe. La opción [/main](#) permite especificar la clase que contiene el método main, en casos en los que el código tiene más de una clase con un método main.

**/t:w** es la forma breve de **/target:winexe**.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades comunes**.
3. Haga clic en la página de propiedades **General**.
4. Modifique la propiedad **Tipo de resultado**.

## Para establecer esta opción del compilador mediante programación

Vea [OutputType \(Propiedad\)](#).

## Ejemplo

Para compilar `in.jsl` en un programa de Windows, ejecute:

```
vjc /target:winexe in.jsl
```

## Vea también

[/target](#) (Especificar el formato del archivo de resultados) | [Opciones del compilador de Visual J#](#)

# /utf8output (Mostrar mensajes del compilador con UTF-8)

`/utf8output[+ | -]`

donde:

+ | -

De forma predeterminada, la opción **/utf8output-** está activada. Si se desea evitar que se muestre el resultado del compilador utilizando la codificación UTF-8, hay que especificar **/utf8output-**. Especificar **/utf8output** equivale a especificar **/utf8output+**.

## Comentarios

La opción **/utf8output** muestra los resultados del compilador utilizando la codificación UTF-8.

En algunas configuraciones internacionales, los resultados del compilador no se pueden mostrar correctamente en la consola. En estas configuraciones, hay que utilizar **/utf8output** y redirigir los resultados del compilador a un archivo.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Vea también

[Opciones del compilador de Visual J#](#)

## /warn (Especificar el nivel de advertencia)

```
/warn:option
```

donde:

*option*

Nivel de advertencia mínimo que se desea mostrar para la generación. Los valores válidos están comprendidos entre 0 y 4:

Nivel de advertencia	Significado
0	Desactiva la emisión de todos los mensajes de advertencia.
1	Muestra los mensajes de advertencia graves.
2	Muestra advertencias de nivel 1 además de otras advertencias menos graves, como las derivadas de ocultar miembros de clase. Éste es el nivel de advertencias predeterminado en la línea de comandos.
3	Muestra advertencias de nivel 2 además de otras advertencias menos graves, como las derivadas de expresiones que siempre producen verdadero o falso.
4	Muestra advertencias de nivel 3 además de otras advertencias informativas.

### Comentarios

La opción **/warn** especifica el nivel de advertencia que debe mostrar el compilador.

Hay que utilizar **/warnaserror** para tratar todas las advertencias como errores.

La opción **/w** es la forma breve de **/warn**.

### Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique la propiedad **Nivel de advertencia**.

### Para establecer esta opción del compilador mediante programación

Vea [WarningLevel \(Propiedad\)](#).

### Ejemplo

Para compilar `in.js` y conseguir que el compilador muestre sólo las advertencias de nivel 1, ejecute:

```
vjc /warn:1 in.js1
```

### Vea también

[Opciones del compilador de Visual J#](#)

# /warnaserror (Tratar advertencias como errores)

```
/warnaserror[+ | -]
```

donde:

+ | -

De manera predeterminada, está activada la opción **/warnaserror-**, que hace que las advertencias no impidan la generación de un archivo de resultados. La opción **/warnaserror+**, que equivale a **/warnaserror**, hace que las advertencias sean tratadas como errores.

## Comentarios

La opción **/warnaserror** trata todas las advertencias como errores. Los mensajes que, normalmente, se mostrarían como advertencias se muestran como errores y el proceso de generación se detiene (no se generan archivos de resultados).

Utilice [/warn](#) para especificar el nivel de advertencia que debe mostrar el compilador.

El compilador no distingue entre niveles de advertencia si se desea tratar las advertencias como errores.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Generar**.
4. Modifique el valor de la propiedad **Tratar advertencias como errores**.

## Para establecer esta opción del compilador mediante programación

Vea [TreatWarningsAsErrors \(Propiedad\)](#).

## Ejemplo

Para compilar `in.js` y hacer que el compilador no muestre advertencias, ejecute:

```
vjc /warnaserror in.js1
```

## Vea también

[Opciones del compilador de Visual J#](#)



# /win32res (Importar un archivo de recursos Win32)

```
/win32res:filename
```

donde:

*filename*

Archivo de recursos que se desea agregar al archivo de resultados.

## Comentarios

La opción **/win32res** inserta un recurso Win32 en el archivo de resultados. Los archivos de recursos Win32 se pueden crear con el [compilador de recursos](#). El Compilador de recursos se invoca cuando se compila un programa de Visual C++; se crea un archivo .res a partir del archivo .rc.

Un recurso Win32 puede contener información de versión o un mapa de bits (icono) que ayudan a identificar la aplicación en el Explorador de Windows. Si no se especifica la opción **/win32res**, el compilador generará información de versión basada en la versión del ensamblado.

Vea [/linkresource](#) (para hacer referencia) o [/resource](#) (para asociar) un archivo de recursos de .NET Framework.

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

Para compilar `in.jsl` y asociar un archivo de recursos Win32 `rf.res` con el fin de generar `in.exe`, ejecute:

```
vjc /win32res:rf.res in.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# /x (Deshabilitar extensiones de lenguaje)

```
/x:[all | net]
```

donde:

all

Deshabilita todas las extensiones de lenguaje. Esto incluye las extensiones de Microsoft incluidas en Visual J++ 6.0 así como las extensiones de lenguaje nuevas específicas de .NET Framework.

net

Deshabilita únicamente las extensiones específicas de .NET Framework. Las extensiones de Microsoft incluidas en Visual J++ 6.0 no se deshabilitan en este modo.

## Comentarios

La opción **/x** deshabilita las extensiones de lenguaje. Para obtener una lista de las extensiones de Microsoft incluidas en Visual J++ 6.0, vea [Compatibilidad con Visual J++ 6.0](#). Para obtener una lista de las extensiones de lenguaje específicas de .NET Framework, vea [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#).

## Para establecer esta opción del compilador en el entorno de desarrollo de Visual Studio

1. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
2. Haga clic en la carpeta **Propiedades de configuración**.
3. Haga clic en la página de propiedades **Avanzadas**.
4. Modifique la propiedad **Opciones adicionales**.

## Para establecer esta opción del compilador mediante programación

Esta opción del compilador no se puede modificar mediante programación.

## Ejemplo

Para compilar `in.jsl` y deshabilitar todas las extensiones de lenguaje, ejecute:

```
vjc /x:all in.jsl
```

## Vea también

[Opciones del compilador de Visual J#](#)

# Opciones del compilador de Visual J++ asignadas a opciones del compilador de Visual J#

Las opciones del compilador de Visual J# difieren de las opciones de compilador que proporciona el compilador de Visual J++® 6.0. En la siguiente tabla se muestran las asignaciones de las opciones de compilador. No todas las opciones del compilador de Visual J++ (jvc.exe) están disponibles en el compilador de Visual J# (vjc.exe). Con la ayuda de esta tabla, se podrán actualizar los archivos MAKE para utilizar el compilador de Visual J#.

Las opciones del compilador de Visual J# no distinguen entre mayúsculas y minúsculas. Algunas de las opciones tienen una forma breve. Para obtener una lista completa de las opciones del compilador de Visual J#, vea

[Opciones del compilador de Visual J# por orden alfabético.](#)

Opción del compilador de Visual J++	Opción del compilador de Visual J#	Comentarios
<b>/?</b>	<b>/?</b> o <b>/help</b>	Muestra información de uso.
<b>/cp &lt;rutadeclase&gt;</b>	<b>/libpath</b>	Se utilizaba para especificar la ruta de la clase para la compilación.  Utilice <b>/libpath</b> o la variable de entorno LIB para especificarlo. Tenga en cuenta que, en tiempo de compilación, se debe hacer referencia también al ensamblado que contiene la clase que utiliza la opción <b>/reference</b> .  <b>/libpath</b> señala a la variable de entorno LIB.
<b>/cp:p[-]</b>	<b>/libpath</b>	Se utilizaba para especificar la ruta de la clase.  Utilice <b>/libpath</b> para indicar la ruta dada en la variable de entorno LIB.
<b>/cp:o[-]</b>	No admitida	Se utilizaba para imprimir la ruta de la clase.
<b>/d &lt;directorio&gt;</b>	No admitida	Se utilizaba para especificar el directorio raíz del resultado del archivo o de clase.  Utilice <b>/out</b> para especificar el directorio del resultado y el nombre del archivo.
<b>/D &lt;símbolo&gt;</b>	<b>/define:&lt;símbolo&gt;</b>	Se utilizaba para definir un símbolo de preprocesador.  Se puede utilizar <b>/define:</b> o su forma breve <b>/d:</b> (o <b>/D:</b> , ya que las opciones de Visual J# .NET no distinguen mayúsculas y minúsculas).
<b>/g</b>	<b>/debug</b>	Se utilizaba para indicar al compilador que generara información de depuración completa.  Utilice la opción <b>/debug</b> en su lugar.
<b>/g:l</b> o <b>/g:t</b>	<b>/debug:{full pdbonly}</b>	Se utilizaba para especificar el tipo de información de depuración que se debía emitir.  Se puede especificar la información de depuración que se debe generar. Especificar <b>/debug:full</b> permite asociar un depurador al programa en ejecución.
<b>/nologo</b>	<b>/nologo</b>	Sin cambios.
<b>/nowarn</b>	<b>/warn:0</b>	Se utilizaba para deshabilitar las advertencias.  Se puede especificar un nivel de advertencia de 0 para deshabilitar todas las advertencias (la forma breve es <b>/w:0</b> ).
<b>/nowrite</b>	No admitida	Se utilizaba para llevar a cabo una compilación sin generar ningún resultado.
<b>/O</b>	<b>/optimize</b>	Se utilizaba para deshabilitar la optimización.  Utilice la opción <b>/optimize</b> en su lugar.
<b>/O:J</b> y <b>/O:I</b>	No admitida	Se utilizaba para especificar el tipo de optimización.

<b>/ref</b>	No admitida	Se utilizaba para compilar de nuevo clases a las que se hacía referencia, si estaban caducadas.
<b>/verbose</b>	No admitida	Se utilizaba para imprimir información sobre el progreso de la compilación.
<b>/w{0-4}</b>	<b>/warn:{0-4}</b>	Se utilizaba para establecer niveles de advertencia.  Ahora se puede utilizar la opción <b>/warn:option</b> (la forma breve es <b>/w:{0-4}</b> ).
<b>/wx</b>	<b>/warnaserror</b>	Se utilizaba para tratar las advertencias como errores.  Hay que utilizar la opción /warnaserror en su lugar.
<b>/x</b>	<b>/x:all</b>	Se utilizaba para deshabilitar las extensiones.  Ahora puede utilizar <b>/x:all</b> para deshabilitar todas las extensiones de lenguaje. <b>/x:net</b> deshabilita únicamente las extensiones específicas de .NET Framework.

## Vea también

[Opciones del compilador de Visual J#](#)

# Errores del compilador: de VJS0001 a VJS9999

Esta sección sirve de referencia para los errores generados por el compilador de Visual J#. Para obtener ayuda acerca de un mensaje de error determinado, haga clic con el *mouse* (ratón) en el número de error que se muestra en la [ventana Resultados](#) y presione F1, o escriba el número de error en el cuadro **Buscar** del Índice.

## Vea también

[Opciones del compilador de Visual J#](#)

# Error del compilador VJS1001

[Compiler Error VJS1001]

## No se proporcionó ningún archivo de código fuente

Se invocó al compilador de Visual J# en la línea de comandos y se especificó una o más opciones del compilador, pero no se especificó ningún archivo de código fuente. Especifique uno o varios archivos de código fuente.

# Error del compilador VJS1002

[Compiler Error VJS1002]

## Opción '*opción*' no reconocida

Se pasó un valor al compilador de Visual J# que comienza con una barra diagonal (/), pero no era una opción del compilador reconocida. Para obtener más información acerca de las opciones del compilador, vea [Opciones del compilador de Visual J#](#).

En el siguiente código se genera el error VJS1002:

```
// VJS1002.js1
// compile with: /nolog
// VJS1002 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1003

[Compiler Error VJS1003]

**La opción '*opción*' debe ir seguida de '+', '-' o nada**

Se pasó un valor no válido a una opción del compilador diseñada para aceptar un valor booleano (o ningún valor).

En el siguiente código se genera el error VJS1003:

```
// VJS1003.js1
// compile with: /optimize:0
// VJS1003 expected
// to resolve, use /optimize (not /optimize:0)
public class MyClass
{
    public static void main()
    {
    }
}
```



# Error del compilador VJS1004

[Compiler Error VJS1004]

**La opción '*opción*' debe ir seguida de ':' y un valor**

Se especificó una opción del compilador, pero mal formada. Vea [Opciones del compilador de Visual J#](#) para obtener información acerca de la sintaxis de las opciones del compilador.

En el siguiente código se genera el error VJS1004:

```
// VJS1004.js1
// compile with: /define+
// VJS1004 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1005

[Compiler Error VJS1005]

## **Archivo '*archivo*' no encontrado**

Se pasó un nombre de archivo al compilador, pero éste no encontró el archivo. El compilador busca los archivos en el directorio actual. Utilice la opción [/recurse](#) del compilador para que éste busque también los archivos en los subdirectorios.

# Error del compilador VJS1006

[Compiler Error VJS1006]

**No se puede leer el archivo de código fuente '*archivo*'**

Se pasó al compilador de Visual J# un archivo de código fuente que no tenía acceso de lectura.

# Error del compilador VJS1009

[Compiler Error VJS1009]

## Archivo `/reference 'archivo'` no encontrado

Se pasó un archivo a la opción del compilador `/reference`, pero el compilador no encontró el archivo.

En el siguiente código se genera el error VJS1009:

```
// VJS1009.js1
// compile with: /reference:a_file_you_do_not_have.dll
// VJS1009 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1010

[Compiler Error VJS1010]

## El archivo */reference 'archivo'* no es un ensamblado

Se pasó un archivo a la opción [/reference](#) del compilador, pero el archivo no contenía un manifiesto de ensamblado.

En el siguiente código se genera el error VJS1010:

```
// VJS1010a.cpp
// compile with: /LD
// a C++ source code file
class SomeClass
{
};
```

```
// VJS1010b.jsl
// compile with: /reference:VJS1010a.dll
// VJS1010 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1011

[Compiler Error VJS1011]

**'@' debe anteceder al nombre del archivo de respuesta**

La opción de la línea de comandos para especificar un archivo de respuesta no incluía el nombre del archivo. Para obtener más información, vea [@ \(especificar archivo de respuesta\)](#).

En el siguiente código se genera el error VJS1011:

```
// VJS1011.js1
// compile with: @
// VJS1011 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1012

[Compiler Error VJS1012]

**No se puede leer el archivo de respuesta '*archivo*'**

Se pasó al compilador de Visual J# un archivo de respuesta que no tenía acceso de lectura.

# Error del compilador VJS1013

[Compiler Error VJS1013]

## El archivo de respuesta '*archivo*' se encontró varias veces

Se especificó un archivo de respuesta dos veces en la misma compilación. Pero los archivos de respuesta sólo se pueden especificar una vez en una compilación. Para obtener más información, vea @ ([especificar archivo de respuesta](#)).

En el siguiente código se genera el error VJS1013:

```
// VJS1013.js1
// compile with: @x.rsp @x.rsp
// VJS1013 expected
public class MyClass
{
    public static void main()
    {
    }
}
```



# Error del compilador VJS1014

[Compiler Error VJS1014]

## Falta la comilla de cierre en el archivo de respuesta

Había un archivo de respuesta que tenía una comilla de apertura, pero no una de cierre. Para el siguiente ejemplo, se supone que en el disco hay un archivo de respuesta con el siguiente contenido:

```
/target:library "
```

En el siguiente código se genera el error VJS1014:

```
// VJS1014.jsl
// compile with: @VJS1014.rsp
// VJS1014 expected
public class MyClass
{
}
```

# Error del compilador VJS1016

[Compiler Error VJS1016]

**'cadena' no es un valor válido para /target**

Se pasó un parámetro no válido a la opción [/target](#) del compilador.

En el siguiente código se genera el error VJS1016:

```
// VJS1016.js1
// compile with: /target:exec
// VJS1016 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1017

[Compiler Error VJS1017]

**'cadena' no es un valor válido para /x**

Se pasó un parámetro no válido a la opción [/x](#) del compilador.

En el siguiente código se genera el error VJS1017:

```
// VJS1017.js1
// compile with: /x:module
// VJS1017 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1019

[Compiler Error VJS1019]

**Valor incorrecto para /baseaddress: '*valor*'**

Se pasó un valor no válido a la opción [/baseaddress](#) del compilador.

En el siguiente código se genera el error VJS1019:

```
// VJS1019.js1
// compile with: /target:library /baseaddress:r
// VJS1019 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1020

[Compiler Error VJS1020]

**El valor de `/warn` debe encontrarse entre 0 y 4**

Se pasó un valor no válido a la opción `/warn` del compilador.

En el siguiente código se genera el error VJS1020:

```
// VJS1020.js1
// compile with: /warn:5
// VJS1020 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1022

[Compiler Error VJS1022]

**No se puede escribir en el archivo '*archivo*'**

No se pudo crear un archivo de resultados especificado por una compilación porque ya hay un archivo con el mismo nombre en el equipo. Cambie el nombre del archivo de resultados o cierre el archivo que tiene el mismo nombre.

# Advertencia del compilador (nivel 1) VJS1023

[Compiler Warning (level 1) VJS1023]

**Se omite el método main duplicado en '*clase*'; hay que utilizar /main:<nombreclase> para especificar el punto de entrada**

El compilador encontró más de un método main. El primero se utiliza como punto de entrada del programa y los siguientes se omiten.

En el siguiente código se genera el error VJS1023:

```
// VJS1023.js1
// compile with: /W:1
public class MyClass
{
    public static void main()
    {
    }
}

public class MyClass2    // VJS1023
{
    // this main method is ignored
    public static void main()
    {
    }
}
```

# Error del compilador VJS1024

[Compiler Error VJS1024]

## Opción /main incorrecta: la clase '*clase*' no contiene un método main adecuado

Una clase que se especificó con la opción [/main](#) del compilador no contenía un método main que se pudiera utilizar como punto de entrada al programa. Especifique una clase que tenga un método main.

En el siguiente código se genera el error VJS1024:

```
// VJS1024.js1
// compile with: /main:MyClass
public class MyClass // VJS1024, specify /main:MyClass2 instead
{
}

public class MyClass2
{
    public static void main()
    {
    }
}
```



# Error del compilador VJS1025

[Compiler Error VJS1025]

**La clase '*clase*' con nombre que hay en la opción /main no se encuentra en la compilación**

El compilador no encontró en el código fuente una clase que se especificó con la opción `/main` del compilador. Especifique una clase que exista en uno de los archivos de código fuente que va a compilar.

En el siguiente código se genera el error VJS1025:

```
// VJS1025.js1
// compile with: /main:MyClass2
// VJS1025 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1026

[Compiler Error VJS1026]

**/target es 'exe' pero no se encontró ningún método main adecuado durante la compilación**

El compilador no encontró ningún método main que actúe como punto de entrada cuando se ejecuta el programa.

En el siguiente código se genera el error VJS1026:

```
// VJS1026.js1
// compile with: /target:exe
// VJS1026 expected
public class MyClass
{
    // uncomment the following lines to resolve
    // public static void main()
    // {
    // }
}
```

# Error del compilador VJS1027

[Compiler Error VJS1027]

**/target es 'winexe' pero no se encontró un método main adecuado durante la compilación**

El compilador no encontró ningún método main que actúe como punto de entrada cuando se ejecuta el programa.

En el siguiente código se genera el error VJS1027:

```
// VJS1027.js1
// compile with: /target:winexe
// VJS1027 expected
public class MyClass
{
    // uncomment the following lines to resolve
    // public static void main()
    // {
    // }
}
```

# Error del compilador VJS1028

[Compiler Error VJS1028]

**/target debe ser 'exe' o 'winexe' si se especifica /main**

La opción [/target:library](#) del compilador no es compatible con la opción [/main](#) del compilador.

Hay que especificar **/target:exe** o **/target:winexe** para generar un ejecutable, o quitar la opción **/main** si se desea generar una biblioteca.

En el siguiente código se genera el error VJS1028:

```
// VJS1028.jsl
// compile with: /target:library /main:MyClass
// VJS1028 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1029

[Compiler Error VJS1029]

**Sólo se puede especificar un identificador de recurso en la opción `/resource` o `/linkresource` de la línea de comandos**

Tanto la opción [/linkresource](#) como la opción [/resource](#) del compilador permiten especificar el nombre del identificador del recurso en el ensamblado. Sin embargo, sólo se puede especificar un nombre de identificador de recurso.

En el siguiente código se genera el error VJS1029:

```
// VJS1029.js1
// compile with: /res:VJS1029.resources,name1,name2
// VJS1029 expected
// try /res:VJS1029.resources,name1 instead
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1030

[Compiler Error VJS1030]

'*archivo*' es un archivo binario y no un archivo de código fuente

Se pasó un archivo binario (por ejemplo, un archivo .dll) al compilador como si fuera un archivo de código fuente.

# Error del compilador VJS1031

[Compiler Error VJS1031]

## Advertencias tratadas como errores

Se utilizó la opción [/warnaserror](#) durante la compilación y las advertencias se tratarán como errores.

En el siguiente código se genera el error VJS1031:

```
// VJS1031.js1
// compile with: /warnaserror /W:3
// VJS1031 expected
class MyClass
{
    public static void main()
    {
        int i;    // warning
    }
}
```

# Error del compilador VJS1032

[Compiler Error VJS1032]

**'valor' no es un valor válido para /debug: sólo se permiten los valores full o pdbonly**

La opción [/debug](#) del compilador se especificó de forma incorrecta.

En el siguiente código se genera el error VJS1032:

```
// VJS1032.js1
// compile with: /debug:0
// VJS1032 expected
// to resolve, use /debug (not /debug:0)
public class MyClass
{
}
```



# Error del compilador VJS1033

[Compiler Error VJS1033]

**Hay que especificar un nombre de archivo de recursos con la opción `/resource` o `/linkresource` de la línea de comandos**

La opción `/resource` o `/linkresource` del compilador se especificó de forma incorrecta.

En el siguiente código se genera el error VJS1033:

```
// VJS1033.js1
// compile with: /resource:
// VJS1033 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1034

[Compiler Error VJS1034]

**'*número*' no es un número de advertencia válido para la opción de la línea de comandos /nowarn.**

Se pasó un valor a la opción [/nowarn](#) del compilador que no era un número de advertencia válido.

En el siguiente código se genera el error VJS1034:

```
// VJS1034.js1
// compile with: /nowarn:0000 /target:library
// VJS1034 expected
public class MyClass
{
}
```

# Error del compilador VJS1035

[Compiler Error VJS1035]

**Falta la especificación de archivo para la opción de la línea de comandos '*opción*'**

La opción del compilador espera una especificación de archivo.

En el siguiente código se genera el error VJS1035:

```
// VJS1035.js1
// compile with: /recurse: /target:library
// VJS1035 expected
// try the following compiler options instead
// /recurse:*.js1 /target:library
public class MyClass
{
}
```

# Error del compilador VJS1036

[Compiler Error VJS1036]

**El identificador de recursos '*nombre*' ya se ha utilizado en este ensamblado**

Ya se ha utilizado en el ensamblado un nombre definido por el usuario para un archivo de recursos.

En el siguiente código se genera el error VJS1036:

```
// VJS1036.jsl
// compile with: /res:VJS1036A.resources,a /res:VJS1036B.resources,a
// VJS1036 expected
public class MyClass
{
}
}
```

# Error del compilador VJS1037

[Compiler Error VJS1037]

**Se especificaron opciones incompatibles: tratar advertencias como errores y nivel de advertencia 0**

No se pueden especificar [/warnaserror](#) y [/warn:0](#) en la misma compilación.

En el siguiente código se genera el error VJS1037:

```
// VJS1037.js1
// compile with: /W:0 /warnaserror
// VJS1037 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1038

[Compiler Error VJS1038]

**La página de códigos '*página de códigos*' no es válida o no está instalada en el sistema**

Se especificó un valor incorrecto para la opción [/codepage](#) al invocar al compilador de Visual J#.

# Error del compilador VJS1039

[Compiler Error VJS1039]

**No se puede vincular el archivo de recursos cuando se genera un módulo**

No se pueden especificar [/target:module](#) ([Crear un módulo para agregarlo a un ensamblado](#)): y [/linkresource](#) ([Vincular a recursos de .NET Framework](#)) en la misma compilación.

# Error del compilador VJS1040

[Compiler Error VJS1040]

**No se puede escribir en el archivo '*nombre de archivo*': '*motivo*'**

El compilador no puede escribir en el archivo por el motivo dado. Por ejemplo, se producirá este error si se abre un archivo en el [Desensamblador MSIL \(Ildasm.exe\)](#) y trata de volver a generar el archivo con vjc.exe.



# Error del compilador VJS1041

[Compiler Error VJS1041]

**Debe especificar un valor para la opción de la línea de comandos '*opción*'**

Se especificó una opción del compilador, pero mal formada. Vea [Opciones del compilador de Visual J#](#) para obtener información acerca de la sintaxis de las opciones del compilador.

En el siguiente código se genera el error VJS1041:

```
// VJS1041.js1
// compile with: /define:
// VJS1041 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1077

[Compiler Error VJS1077]

## Comentario sin terminar

No se terminó un comentario de varias líneas.

En el siguiente código se genera el error VJS1077:

```
// VJS1077.js1
class MyClass
{
/*    // VJS1077, no end to comment
    public static void main()
    {
        System.out.println("test");
    }
}
```

# Error del compilador VJS1078

[Compiler Error VJS1078]

## Carácter inesperado '*carácter*'

El compilador detectó un error de sintaxis debido a un carácter inesperado.

En el siguiente código se genera el error VJS1078:

```
// VJS1078.js1
class MyClass
{
    @    // VJS1078
}
```

# Error del compilador VJS1079

[Compiler Error VJS1079]

**'*símbolo*' no es una palabra clave ni un identificador válido**

Visual J# no reconoce **const** ni **goto** como identificadores válidos.

En el siguiente código se genera el error VJS1079:

```
// VJS1079.js1
class MyClass
{
    int goto;    // VJS1079
}
```

# Error del compilador VJS1080

[Compiler Error VJS1080]

***#error 'error definido por el usuario'***

Este error se genera cuando se ejecuta una directiva **#error**.

En el siguiente código se genera el error VJS1080:

```
// VJS1080.js1
public class MyClass
{
    public static void main()
    {
        #error A user-defined error    // VJS1080
    }
}
```

# Advertencia del compilador (nivel 1) VJS1081

[Compiler Warning (level 1) VJS1081]

**#warning** '*advertencia definida por el usuario*'

Esta advertencia se genera cuando se ejecuta una directiva **#warning**.

En el siguiente código se genera el error VJS1081:

```
// VJS1081.js1
// compile with: /W:1
public class MyClass
{
    public static void main()
    {
        #warning A user-defined warning    // VJS1081
    }
}
```

# Error del compilador VJS1082

[Compiler Error VJS1082]

**Las directivas condicionales deben aparecer en primer lugar en una línea sin espacios en blanco**

En una línea había texto delante de una directiva condicional. Siempre hay que especificar las directivas condicionales como primer símbolo de una línea.

En el siguiente código se genera el error VJS1082:

```
// VJS1082.js1
#define Xc
public class MyClass
{
    public static void main()
    {
        int i; #if X // VJS1082
            int i;
        #endif
    }
}
```

# Error del compilador VJS1083

[Compiler Error VJS1083]

**Se esperaba un identificador condicional, pero se encontró '*cadena*'**

El compilador detectó un símbolo mal formado en una directiva condicional.

En el siguiente código se genera el error VJS1083:

```
// VJS1083.js1
#define -X    // VJS1083
#define VALID_ID    // OK
class MyClass
{
}
```



# Error del compilador VJS1084

[Compiler Error VJS1084]

**Se esperaba una directiva condicional, pero se encontró '*cadena*'**

El compilador detectó una directiva condicional mal formada.

En el siguiente código se genera el error VJS1084:

```
// VJS1084.js1
#define X    // VJS1084
class MyClass
{
}
```

# Error del compilador VJS1085

[Compiler Error VJS1085]

## Se esperaba un identificador condicional

El compilador detectó una directiva condicional mal formada.

En el siguiente código se genera el error VJS1085:

```
// VJS1085.js1
#define    // VJS1085
class MyClass
{
}
```

# Error del compilador VJS1086

[Compiler Error VJS1086]

**Se esperaba una directiva condicional, pero se encontró el final de la línea**

Se encontró el final de la línea en lugar de un identificador.

En el siguiente código se genera el error VJS1086:

```
// VJS1086.js1
class MyClass
{
    #    // VJS1086
    #if DIRECTIVE    // OK
    #endif
}
```

# Error del compilador VJS1087

[Compiler Error VJS1087]

**La directiva #line debe ir seguida de un número de línea o de 'default'**

El compilador encontró una directiva `#line` incompleta.

En el siguiente código se genera el error VJS1087:

```
// VJS1087.js1
public class MyClass
{
    public static void main()
    {
        #line testing    // VJS1087
        // try the following line instead
        // #line 21
    }
}
```

# Error del compilador VJS1088

[Compiler Error VJS1088]

**'*#text*' no es una directiva condicional**

El compilador encontró un símbolo que comenzaba con el signo #, pero que no era una directiva condicional válida.

En el siguiente código se genera el error VJS1088:

```
// VJS1088.js1
public class MyClass
{
    public static void main()
    {
        #linee 22 // VJS1088
        // try the following line instead
        // #line 22
    }
}
```

# Error del compilador VJS1089

[Compiler Error VJS1089]

## Texto adicional después de una directiva condicional

Se encontró texto adicional después de una directiva condicional que, de no ser por este texto, estaría bien formada.

En el siguiente código se genera el error VJS1089:

```
// VJS1089.js1
public class MyClass
{
    public static void main()
    {
        #line 22 77    // VJS1089
    }
}
```

# Error del compilador VJS1090

[Compiler Error VJS1090]

## No puede haber **#define** o **#undef** después de código fuente

El compilador encontró una directiva **#define** o **#undef** después del inicio del código fuente del archivo. Estas directivas se deben especificar delante del primer símbolo del código fuente.

En el siguiente código se genera el error VJS1090:

```
// VJS1090.js1
import System.*;
#define DEBUG // VJS1090, try moving this line before import statement
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1091

[Compiler Error VJS1091]

'**directiva**' no tiene su **#if** correspondiente

Se encontró una directiva condicional, **#endif** o **#else**, sin una directiva **#if** correspondiente.

En el siguiente código se genera el error VJS1091:

```
// VJS1091.js1
public class MyClass
{
    public static void main()
    {
        // #if DEBUG
        #endif    // VJS1091, uncomment the #if statement to resolve
    }
}
```



# Error del compilador VJS1092

[Compiler Error VJS1092]

## **#if no tiene su #endif correspondiente**

Se encontró una directiva **#if** sin una directiva **#endif** correspondiente.

En el siguiente código se genera el error VJS1092:

```
// VJS1092.js1
public class MyClass
{
    public static void main()
    {
        #if DEBUG    // VJS1092
        // uncomment the following line to resolve
        // #endif
    }
}
```

# Error del compilador VJS1093

[Compiler Error VJS1093]

**'#if' tiene varias '#else'**

Una instrucción **#if** sólo puede tener una directiva **#else**.

En el siguiente código se genera el error VJS1093:

```
// VJS1093.js1
public class MyClass
{
    public static void main()
    {
        #if DEBUG
            // DEBUG defined
        #else
            // DEBUG not defined
        #else    // VJS1093, second #else not valid
            // something wrong
        #endif
    }
}
```

# Error del compilador VJS1094

[Compiler Error VJS1094]

## Final de línea inesperado en una expresión condicional

Había una directiva condicional mal formada.

En el siguiente código se genera el error VJS1094:

```
// VJS1094.js1
public class MyClass
{
    public static void main()
    {
        #if    // VJS1094, add an expression to evaluate
        // try the following line instead
        // #if DEBUG
        #endif

    }
}
```

# Error del compilador VJS1095

[Compiler Error VJS1095]

## Falta '(' en una expresión condicional

Había una expresión condicional mal formada porque faltaba un paréntesis de apertura.

En el siguiente código se genera el error VJS1095:

```
// VJS1095.js1
public class MyClass
{
    public static void main()
    {
        #if (DEBUG    // VJS1095
        // try the following line instead
        // #if (DEBUG)
        #endif
    }
}
```

# Error del compilador VJS1096

[Compiler Error VJS1096]

## No se puede cambiar un símbolo 'true' o 'false' predefinido

El compilador detectó un intento de definir un símbolo mediante una constante booleana: true o false. No se puede modificar el significado de true o false.

En el siguiente código se genera el error VJS1096:

```
// VJS1096.js1
#define true    // VJS1096, don't use true as a user-defined symbol
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1097

[Compiler Error VJS1097]

**La directiva '#directiva' no se admite cuando las extensiones de .NET están deshabilitadas**

Algunas directivas de procesamiento no son válidas cuando están deshabilitadas las extensiones de .NET Framework. Para obtener más información, vea la opción [/x](#) del compilador.

En el siguiente código se genera el error VJS1097:

```
// VJS1097.js1
// compile with: /x:net
public class MyClass
{
    public static void main()
    {
        #line 32    // VJS1097
    }
}
```

# Error del compilador VJS1118

[Compiler Error VJS1118]

**No se esperaba '*símbolo*'**

Se encontró un símbolo (token) inesperado.

En el siguiente código se genera el error VJS1118:

```
// VJS1118.js1
public class MyClass
{
    public static void main(String[] args)
    {
        )    // VJS1118
    }
}
```

# Error del compilador VJS1119

[Compiler Error VJS1119]

**Se esperaba '*símbolo*'**

El compilador no pudo analizar el código fuente.

En el siguiente código se genera el error VJS1119:

```
// VJS1119.js1
package test
public class MyClass // VJS1119, end previous line with semicolon
{
    public static void main()
    {
    }
}
```



# Error del compilador VJS1120

[Compiler Error VJS1120]

**Se esperaba 'class', 'interface' o 'delegate', pero se encontró '*símbolo*'**

El compilador detectó una declaración no válida.

En el siguiente código se genera el error VJS1120:

```
// VJS1120.js1
class MyClass
{
    public void Test()
    {
        abstract int t;    // VJS1120
        int x = 10;        // OK
    }
}
```

# Error del compilador VJS1122

[Compiler Error VJS1122]

**Se esperaba type, pero se encontró 'void'**

**void** no es un tipo válido para parámetros de una definición o declaración de método.

En el siguiente código se genera el error VJS1122:

```
// VJS1122.js1
class MyClass
{
    public void Test(void t)    // VJS1122
    // try the following line instead
    // public void Test(int t)
    {
    }
}
```

# Error del compilador VJS1123

[Compiler Error VJS1123]

## Declaración de miembro incorrecta

No se declaró correctamente un miembro de clase.

En el siguiente código se genera el error VJS1123:

```
// VJS1123.js1
public class MyClass
{
    public static void main()
    {
    }
    xxx class MyClass2    // VJS1123, xxx is not a valid access modifier
    // try the following line instead
    // public class MyClass2
    {
    }
}
```

# Error del compilador VJS1124

[Compiler Error VJS1124]

**@attribute.return sólo es válido en declaraciones de método**

El atributo **@attribute.return** se aplicó a una construcción para la que no estaba diseñado; sólo se puede aplicar a métodos.

En el siguiente código se genera el error VJS1124:

```
// VJS1124.js1
import System.Runtime.InteropServices.*;
/** @attribute.return MarshalAs(UnmanagedType.Error) */ // VJS1124
public class MyClass
{
    // OK
    /** @attribute.return MarshalAs(UnmanagedType.Error) */
    public int Test()
    {
        return -1;
    }
}
```

# Error del compilador VJS1125

[Compiler Error VJS1125]

## Final inesperado de una declaración de atributo

Hay una declaración de atributo que no está completa.

En el siguiente código se genera el error VJS1125:

```
// VJS1125.js1
/** @attribute System.CLSCompliant */    // VJS1125
// try the following line instead
// /** @attribute System.CLSCompliant(true) */
class MyClass
{
}
```

# Error del compilador VJS1126

[Compiler Error VJS1126]

**@attribute.method sólo es válido en declaraciones de método**

El atributo **@attribute.method** se aplicó a una construcción para la que no estaba diseñado; sólo se puede aplicar a métodos.

En el siguiente código se genera el error VJS1126:

```
// VJS1126.js1
/** @attribute.method System.ObsoleteAttribute("Hello") */    // VJS1126
public class MyClass
{
    // following line is OK
    /** @attribute.method System.ObsoleteAttribute("Hello") */
    public static void main()
    {
    }
}
```

# Error del compilador VJS1127

[Compiler Error VJS1127]

## No se permiten constructores definidos por el usuario en una clase marcada con **ComImportAttribute**

No se puede especificar un constructor definido por el usuario en una clase marcada con **ComImportAttribute**. La capa de interoperabilidad COM de Common Language Runtime proporciona el constructor para las clases **ComImport**. Este constructor es necesario para que un objeto COM parezca un objeto administrado en tiempo de ejecución.

En el siguiente código se genera el error VJS1127:

```
// VJS1127.jsl
import System.Runtime.InteropServices.*;
/** @attribute ComImport() */
/** @attribute Guid("fb6363f0-721b-478e-bfc6-bc30f7b06be5") */
public class MyClass // VJS1127
{
    // To solve the problem, remove the constructor below.
    public MyClass(){}
}
```

# Error del compilador VJS1128

[Compiler Error VJS1128]

**Una clase marcada con ComImportAttribute debe tener GuidAttribute**

**GuidAttribute** debe estar preestablecido en una clase marcada con **ComImportAttribute**. De lo contrario, la capa de interoperabilidad COM de .NET Framework no puede crear una clase COM.

En el siguiente código se genera el error VJS1128:

```
// VJS1128.jsl
// compile with: /target:library
import System.Runtime.InteropServices.*;
/** @attribute ComImport() */
// add the following line to resolve
// /** @attribute Guid("00000000-0000-0000-0000-000000000001") */
public class MyClass    // VJS1128
{
}
```



# Error del compilador VJS1129

[Compiler Error VJS1129]

**El campo de instancia '*campo*' de una clase marcada con `StructLayoutAttribute(LayoutKind.Explicit)` debe tener `FieldOffsetAttribute`**

Cuando [StructLayout Explicit](#) está activo, los campos deben estar marcados con [FieldOffset](#).

En el siguiente código se genera el error VJS1129:

```
// VJS1129.jsl
// compile with: /target:library
import System.Runtime.InteropServices.*;

/** @attribute StructLayout(LayoutKind.Explicit, Size = 4)*/
class Myclass
{
    int a;
}
```

# Error del compilador VJS1153

[Compiler Error VJS1153]

**La clase '*clase*' se hereda de sí misma (circularidad)**

Una clase no se puede utilizar a sí misma como clase base.

En el siguiente código se genera el error VJS1153:

```
// VJS1153.js1
public class MyClass extends MyClass    // VJS1153
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1156

[Compiler Error VJS1156]

**La superclase '*clase*' es final y no se puede extender**

Una clase marcada como **final** no se puede utilizar como clase base.

En el siguiente código se genera el error VJS1156:

```
// VJS1156.js1
final class MyClass2
{
    public static void main()
    {
    }
}

public class MyClass extends MyClass2    // VJS1156
{
}
```

# Error del compilador VJS1157

[Compiler Error VJS1157]

**la interfaz '*interfaz*' no se puede extender (utilice 'implements' en su lugar)**

Una clase no puede extender una interfaz, pero puede implementarla.

En el siguiente código se genera el error VJS1157:

```
// VJS1157.js1
interface Point
{
}

public class MyClass extends Point    // VJS1157
// try the following line instead
// public class MyClass implements Point
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1158

[Compiler Error VJS1158]

**Una interfaz no puede implementar una interfaz (utilice 'extends' en su lugar)**

Una interfaz no puede implementar otra interfaz, pero puede implementarla.

En el siguiente código se genera el error VJS1158:

```
// VJS1158.js1
interface Point
{
}

interface Point2 implements Point    // VJS1158
// try the following line instead
// interface Point2 extends Point
{
}

public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1159

[Compiler Error VJS1159]

**La interfaz '*interfaz*' se ha implementado dos veces**

Una interfaz sólo se puede implementar una vez por tipo.

En el siguiente código se genera el error VJS1159:

```
// VJS1159.js1
interface Point
{
}

interface Point2 extends Point, Point    // VJS1159, Point listed twice
// try the following line instead
// interface Point2 extends Point
{
}

public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1161

[Compiler Error VJS1161]

**No se puede encontrar la clase '*clase*'**

Se importó un tipo, pero no estaba en la compilación.

En el siguiente código se genera el error VJS1161:

```
// VJS1161a.jsl
// compile with: /target:library
public class Class1
{
}
```

```
// VJS1161b.jsl
// compile with: /target:library
import Class1; // VJS1161, add VJS1161a.jsl to compilation
class MyClass
{
}
```

# Error del compilador VJS1163

[Compiler Error VJS1163]

**'*clase*' no es una interfaz**

Una clase no se puede implementar, pero se puede extender.

En el siguiente código se genera el error VJS1163:

```
// VJS1163.js1
interface Point
{
}

public class MyClass
{
}

public class MyClass2 implements MyClass    // VJS1163
// try the following line instead
// public class MyClass2 extends MyClass
{
    public static void main()
    {
    }
}
```



# Error del compilador VJS1164

[Compiler Error VJS1164]

## No se permiten declaraciones de interfaz en bloques

Se declaró una interfaz en una ubicación no válida.

En el siguiente código se genera el error VJS1164:

```
// VJS1164.js1
// compile with: /target:library
class MyClass
{
    {
        interface I    // VJS1164, interface declaration not allowed here
        {
        }

        class C    // type declarations allowed
        {
        }
    }
}
```

# Error del compilador VJS1166

[Compiler Error VJS1166]

## No se permiten inicializadores de instancia ni estáticos en interfaces

Una interfaz no puede contener bloques de inicializadores de instancia o estáticos, pero puede contener inicializaciones de miembro.

En el siguiente código se genera el error VJS1166:

```
// VJS1166.js1
// compile with: /target:library
interface I1
{
    static    // VJS1166 static initializer block in interface
    {
        int i = 10;
    }

    static int x = 0;    // OK: static member initializer
}

interface I2
{
    {    // VJS1166 instance initializer block in interface
        int i = 10;
    }

    int x = 20;    // OK: instance member initializer
}
```

# Error del compilador VJS1167

[Compiler Error VJS1167]

## No se permiten constructores en interfaces

Una interfaz no puede contener la declaración o definición de un constructor.

En el siguiente código se genera el error VJS1167:

```
// VJS1167
public interface Point
{
    Point()    // VJS1167, constructor not allowed
    {
    }
}

public class MyClass
{
    public MyClass ()    // OK, Ctors allowed in class declarations
    {
    }

    public static void main()
    {
    }
}
```

# Error del compilador VJS1169

[Compiler Error VJS1169]

'**clase**' se debe declarar como abstract o se debe implementar '**clasebase.miembro**'

Una clase (**clase**) debe implementar los miembros abstractos en una clase heredada (**clasebase**), a menos que la **clase** se declare abstracta.

En el siguiente código se genera el error VJS1169:

```
// VJS1169.js1
public interface Point
{
    void SetPoint();
}

public class MyClass implements Point    // VJS1169
{
    // uncomment member definition to resolve, or declare MyClass abstract
    // public void SetPoint()
    // {
    // }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1170

[Compiler Error VJS1170]

'**clase**' se debe declarar como abstract o se deben implementar estos métodos: *métodos*

Una clase (**clase**) debe implementar los miembros abstractos en una clase heredada, a menos que la **clase** se declare abstracta.

En el siguiente código se genera el error VJS1170:

```
// VJS1170.js1
public interface Point
{
    void SetPoint();
    int GetPoint();
}

public class MyClass implements Point // VJS1170
// to resolve, either implement methods in Point or
// declare MyClass as abstract
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1172

[Compiler Error VJS1172]

**Es necesaria una instancia envolvente explícita de la clase '*clase*' para crear una instancia de la clase interna '*claseinterna*'**

Se requiere una instancia de objeto de clase envolvente para crear una instancia de un miembro de clase interna no estática.

En el siguiente código se genera el error VJS1172:

```
// VJS1172.js1
class MyClass
{
    public static void main (String [] args)
    {
        new Class1.InnerClass();    // VJS1172
        // to resolve, define InnerClass as static
        // or try the following line instead
        // new Class1 (). new InnerClass ();
    }
}

class Class1
{
    public class InnerClass
    {
    }
}
```

# Error del compilador VJS1173

[Compiler Error VJS1173]

**Es necesaria una instancia envolvente explícita de la clase '*clase*' para llamar al constructor de la clase interna '*claseinterna*'**

Se intentó crear una clase interna a partir de un contexto estático; se necesita una instancia para crear la clase interna.

En el siguiente código se genera el error VJS1173:

```
// VJS1173a.jsl
// compile with: /target:library
class MyClass
{
    public class InnerClass
    {
    }
    public static class Inner2 extends InnerClass
    {
        Inner2 ()
        {
            super();    // VJS1173, make Inner2 an instance class
        }
    }
}
```

Cada constructor de clase tiene una llamada a su superclase, por lo que se necesita una instancia.

En el siguiente código se genera el error VJS1173:

```
// VJS1173b.jsl
// compile with: /target:library
public class MyClass extends Class1.Inner
{
    public MyClass()    // VJS1173, Class1.Inner needs to be static
    {
    }
}

public class Class1
{
    public class Inner
    {
    }
}
```

# Error del compilador VJS1174

[Compiler Error VJS1174]

**La clase tiene el mismo nombre sencillo que la clase envolvente '*clase*'**

Una clase interna tiene el mismo nombre que su clase externa.

En el siguiente código se genera el error VJS1174:

```
// VJS1174.js1
public class MyClass
{
    public static void main()
    {
    }
    public class MyClass    // VJS1174
    // try the following line instead
    // public class InnerClass
    {
    }
}
```



# Error del compilador VJS1175

[Compiler Error VJS1175]

## La clase interna '*clase*' contiene un inicializador estático

Una clase interna de instancia no puede contener un inicializador estático, mientras que una clase interna estática sí puede contener un inicializador estático.

En el siguiente código se genera el error VJS1175:

```
// VJS1175.js1
public class MyClass
{
    public static void main()
    {
    }

    public class InnerClass    // VJS1175
    {
        static int i = 0;
        // try the following line instead
        // int i = 0;
    }
}
```

# Error del compilador VJS1176

[Compiler Error VJS1176]

## No se puede declarar el campo '*campo*' como static en una clase interna

Una clase interna de instancia no puede contener un campo estático, mientras que una clase interna estática sí puede contener declaraciones de campo estático.

En el siguiente código se genera el error VJS1176:

```
// VJS1176.js1
public class MyClass
{
    public static void main()
    {
    }

    public class InnerClass
    {
        static int i = 0;    // VJS1176
        // try the following line instead
        // int i = 0;
    }
}
```

# Error del compilador VJS1177

[Compiler Error VJS1177]

## No se puede declarar el método '*método*' como static en una clase interna

Una clase interna de instancia no puede contener un método estático, mientras que una clase interna estática sí puede contener declaraciones de método estático.

En el siguiente código se genera el error VJS1177:

```
// VJS1177.js1
public class MyClass
{
    public static void main()
    {
    }

    public class InnerClass
    {
        public static int Test()    // VJS1177, delete static modifier
        {
        }
    }
}
```

# Error del compilador VJS1179

[Compiler Error VJS1179]

**No se puede declarar la interfaz '*interfaz*' en una clase interna**

Una clase interna no estática no puede contener interfaces.

En el siguiente código se genera el error VJS1179:

```
// VJS1179.js1
class MyClass
{
    public class Inner
    {
        public interface InnerInterface    // VJS1179, to resolve, declare Inner as static
        {
        }
    }
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1180

[Compiler Error VJS1180]

**La expresión 'new' cualificada debe utilizar un nombre de tipo sencillo, no '*nombre*'**

Cuando se crea un objeto desde una clase interna, el tipo interno debe ser un nombre de tipo sencillo.

En el siguiente código se genera el error VJS1180:

```
// VJS1180.js1
class MyClass
{
    public static void main(String [] args)
    {
        new Class1 ().new Class1.Inner();    // VJS1180
        // try the following line instead
        // new Class1 (). new Inner();
    }
}

class Class1
{
    public class Inner
    {
    }
}
```

# Error del compilador VJS1182

[Compiler Error VJS1182]

**No se puede asignar 'this' a '*clase*' para una instancia envolvente para '*claseinterna*'**

Para poder crear la instancia de una clase interna, se necesita una instancia de la clase envolvente de la clase interna.

En el siguiente código se genera el error VJS1182:

```
// VJS1182.js1
class MyClass
{
    public void Test()
    {
        new Class1.Inner();    // VJS1182
        // try the following line instead
        // new Class1 (). new Inner ();
    }
    public static void main (String [] args)
    {
    }
}

class Class1
{
    public class Inner
    {
    }
}
```

# Error del compilador VJS1183

[Compiler Error VJS1183]

**El tipo '*tipo*' declarado en otro archivo de código fuente se ha declarado de nuevo**

Se declaró un tipo con el mismo nombre en dos archivos de código fuente de la misma compilación.

En el siguiente código se genera el error VJS1183:

```
// VJS1183a.jsl
// compile with: /target:library
public class MyClass
{
}
```

```
// VJS1183b.jsl
// compile with: VJS1183a.jsl
public class MyClass    // VJS1183, MyClass also declared in VJS1183a.jsl
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1184

[Compiler Error VJS1184]

**El tipo '*tipo*' declarado en el mismo archivo de código fuente se ha declarado de nuevo**

Un tipo con el mismo nombre se declaró dos veces en un archivo de código fuente.

En el siguiente código se genera el error VJS1184:

```
// VJS1184.js1
public class MyClass
{
    public static void main()
    {
    }
}

class MyClass    // VJS1184
{
}
```



# Error del compilador VJS1185

[Compiler Error VJS1185]

**Al menos dos instrucciones 'import on demand' importaron el tipo '*tipo*'**

El compilador detectó que al menos dos instrucciones **import** importaron un nombre sencillo de tipo que se encuentra en dos paquetes diferentes.

En el siguiente código se genera el error VJS1185:

```
// VJS1185a.jsl
// compile with: /target:library
package pkg1;
public class MyType
{
}
```

```
// VJS1185b.jsl
// compile with: /target:library
package pkg2;
public class MyType
{
}
```

```
// VJS1185c.jsl
// compile with: VJS1185a.jsl VJS1185b.jsl
// VJS1185 expected
import pkg1.*;
import pkg2.*;

class MyClass
{
    MyType t;
}
```

# Advertencia del compilador (nivel 1) VJS1186

[Compiler Warning (level 1) VJS1186]

**'import' y java.lang importaron el tipo '*tipo*'. Se utilizará el tipo java.lang.**

El paquete java.lang se importa de forma predeterminada. Esta advertencia indica que una clase del paquete java.lang está definida en otro paquete y se importa mediante la instrucción **import**.

En el siguiente código se genera el error VJS1186:

```
// VJS1186.js1
// compile with: /W:1
// VJS1186 expected
import System.*;
class MyClass
{
    public static void main (String [] args)
    {
        Exception e; // VJS1186: Exception is in java.lang package and .NET framework System p
ackage
    }
}
```

# Error del compilador VJS1190

[Compiler Error VJS1190]

**El campo '*campo*' es ambiguo, se hereda de varias superclases o superinterfaces**

Se debe calificar el uso de un campo en una clase derivada.

En el siguiente código se genera el error VJS1190:

```
// VJS1190.js1
interface I1
{
    int i = 0;
}

interface I2
{
    int i = 0;
}

public class MyClass implements I1, I2
{
    public static void main()
    {
        int j = i;    // VJS1190
        // try the following line instead
        // int j = I1.i;
    }
}
```

# Error del compilador VJS1191

[Compiler Error VJS1191]

**El campo '*campo*' ya está definido en la clase actual**

Se definió un campo más de una vez en una clase.

En el siguiente código se genera el error VJS1191:

```
// VJS1191.js1
public class MyClass
{
    int i;
    int i;    // VJS1191, delete line or rename identifier
    public static void main()
    {
    }
}
```

# Error del compilador VJS1192

[Compiler Error VJS1192]

**El campo final en blanco '*campo*' no está inicializado en la declaración ni en el constructor**

Un campo marcado como **final** debe inicializarse cuando se declara o en el constructor.

En el siguiente código se genera el error VJS1192:

```
// VJS1192.js1
public class MyClass    // VJS1192
{
    final int i;
    // try the following line instead
    // final int i = 0;
    public static void main()
    {
    }
}
```

# Error del compilador VJS1193

[Compiler Error VJS1193]

## El campo final '*campo*' se inicializó más de una vez

Un campo **final** no se puede inicializar más de una vez. El compilador detectó que un campo **final** se inicializó en el bloque de inicialización de la instancia y en el constructor.

En el siguiente código se genera el error VJS1193:

```
// VJS1193.js1
class MyClass    // VJS1193, delete one of the initializations
{
    final int i;
    {
        i = 10;
    }
    MyClass ()
    {
        i = 20;
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1194

[Compiler Error VJS1194]

**No se puede asignar una variable o campo final '*campo*'**

Se inicializó un campo **final** en el constructor después de haberse inicializado en la declaración.

En el siguiente código se genera el error VJS1194:

```
// VJS1194.js1
class MyClass
{
    final int i = 20;
    MyClass ()
    {
        i = 10;    // VJS1194
    }
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1195

[Compiler Error VJS1195]

## El campo final en blanco '*campo*' ya se ha inicializado

Un campo **final** se inicializó dos veces en el constructor.

En el siguiente código se genera el error VJS1195:

```
// VJS1195.js1
class MyClass
{
    final int i;
    MyClass ()
    {
        i = 10;
        i = 20;    // VJS1195, delete one of the initializations
    }
    public static void main()
    {
    }
}
```



# Error del compilador VJS1196

[Compiler Error VJS1196]

**El campo final en blanco '*campo*' no se puede inicializar dentro de una instrucción de bucle**

Apareció la inicialización de un campo **final** en un bucle. Un campo **final** sólo se puede inicializar una vez.

En el siguiente código se genera el error VJS1196:

```
// VJS1196.js1
class MyClass
{
    final int i;
    {
        int k = 1;
        while (k == 1)
        {
            i = 10;    // VJS1196, in a loop
        }
        i = 10;
    }

    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1197

[Compiler Error VJS1197]

**El lado izquierdo de una asignación debe ser un valor l**

Había una asignación mal formada.

En el siguiente código se genera el error VJS1197:

```
// VJS1197.js1
public class MyClass
{
    public static void main()
    {
        int i = 9;
        0 = i;    // VJS1197
        // try the following line instead
        // i = 0;
    }
}
```

# Error del compilador VJS1200

[Compiler Error VJS1200]

**El tipo '*tipo*' ya es un tipo conocido en el contexto actual**

Se declararon dos tipos con el mismo nombre en el mismo ámbito.

En el siguiente código se genera el error VJS1200:

```
// VJS1200.js1
class MyClass
{
    public class Type1
    {
    }
    public class Type1    // VJS1200, second type with same name
    {
    }
}
```

# Error del compilador VJS1201

[Compiler Error VJS1201]

**El nombre '*identificador*' no está definido en el contexto actual**

Se utilizó un identificador no definido en el ámbito actual.

En el siguiente código se genera el error VJS1201:

```
// VJS1201.js1
public class MyClass
{
    public static void main()
    {
        // int ii = 0;
        ii++;    // VJS1201, uncomment previous line to resolve
    }
}
```

# Error del compilador VJS1202

[Compiler Error VJS1202]

**La variable '*variable*' ya está definida en el bloque actual**

Se declaró dos veces una variable con el mismo nombre.

En el siguiente código se genera el error VJS1202:

```
// VJS1202.js1
public class MyClass
{
    public static void main()
    {
        int i;
        int i;    // VJS1202, delete or rename
    }
}
```

# Error del compilador VJS1203

[Compiler Error VJS1203]

**Debe inicializar la variable local '*variable*' antes de usarla**

Las variables deben inicializarse para que se puedan utilizar.

En el siguiente código se genera el error VJS1204:

```
// VJS1203.js1
public class MyClass
{
    public static void main()
    {
        char c;
        // try the following line instead
        // char c = 'a';
        if (c == 'a')    // VJS1203
            ;
    }
}
```

# Error del compilador VJS1204

[Compiler Error VJS1204]

**La condición if debe ser booleana y no '*tipo*'**

La parte condicional de una instrucción **if** no era una expresión booleana.

En el siguiente código se genera el error VJS1204:

```
// VJS1204.js1
public class MyClass
{
    public static void main()
    {
        char c = 'a';
        if ('a')    // VJS1204
        // try the following line instead
        // if (c == 'a')
            ;
    }
}
```

# Error del compilador VJS1205

[Compiler Error VJS1205]

**La condición for debe ser booleana y no '*tipo*'**

La parte condicional de una instrucción **for** no era una expresión booleana.

En el siguiente código se genera el error VJS1205:

```
// VJS1205.js1
public class MyClass
{
    public static void main()
    {
        for (int i = 0 ; 'c' ; i++)    // VJS1205
            // try the following line instead
            // for (int i = 0 ; i < 10 ; i++)
                ;
    }
}
```



# Error del compilador VJS1206

[Compiler Error VJS1206]

**La condición while debe ser booleana y no '*tipo*'**

La parte condicional de una instrucción **while** no era una expresión booleana.

En el siguiente código se genera el error VJS1206:

```
// VJS1206.js1
public class MyClass
{
    public static void main()
    {
        char c = 'a';
        while ('c')    // VJS1206
        // try the following line instead
        // while (c != 'c')
            ;
    }
}
```

# Error del compilador VJS1207

[Compiler Error VJS1207]

**La condición `do` debe ser booleana y no '*tipo*'**

La instrucción condicional de un bucle **`do`** debe ser de tipo **`boolean`**.

En el siguiente código se genera el error VJS1207:

```
// VJS1207.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        do
        {
            i++;
        } while ('c');    // VJS1207
        // try the following line instead
        // } while (i < 10);
    }
}
```

# Error del compilador VJS1208

[Compiler Error VJS1208]

**La etiqueta '*etiqueta*' ya está definida en la jerarquía de instrucciones actual**

Se declaró un nombre de etiqueta en el ámbito de una etiqueta con el mismo nombre.

En el siguiente código se genera el error VJS1208:

```
// VJS1208.js1
class MyClass
{
    public static void main()
    {
        int i = 0;
        label1:
        {
            i = 1;
            label1:    // VJS1208, already in label1 scope
            {
                i = 1;
            }
        }
    }
}
```

# Error del compilador VJS1209

[Compiler Error VJS1209]

**La etiqueta '*etiqueta*' no está definida en la jerarquía de instrucciones actual**

Se llamó a una etiqueta pero no se declaró.

En el siguiente código se genera el error VJS1209:

```
// VJS1209.js1
class MyClass
{
    public static void main (String [] args)
    {
        for ( int i = 0 ; i < 10 ; i++ )
        {
            continue MyLabel;    // VJS1209, no such label declared
        }
    }
}
```

# Error del compilador VJS1210

[Compiler Error VJS1210]

**Este selector predeterminado debe ser exclusivo en la instrucción switch**

Se encontró más de una condición predeterminada en una instrucción **switch**.

En el siguiente código se genera el error VJS1210:

```
// VJS1210.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            case 0:
            default:
            default:    // VJS1210
        }
    }
}
```

# Error del compilador VJS1211

[Compiler Error VJS1211]

**La etiqueta '*etiqueta*' ya está definida en la etiqueta switch**

El compilador encontró dos instrucciones case con valores duplicados.

En el siguiente código se genera el error VJS1211:

```
// VJS1211.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            case 0:
            case 0:    // VJS1211
                // try the following line instead
                // case 1:
        }
    }
}
```

# Error del compilador VJS1212

[Compiler Error VJS1212]

**La etiqueta switch debe ser ordinal y no '*tipo*'**

El valor especificado en una instrucción case no era un ordinal.

En el siguiente código se genera el error VJS1212:

```
// VJS1212.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            case "abc":    // VJS1212, string not an ordinal
                           // try the following line instead
                           // case 0:
            default:
        }
    }
}
```

# Error del compilador VJS1213

[Compiler Error VJS1213]

## La etiqueta switch debe ser un valor constante

El valor pasado a una instrucción case no era una constante.

En el siguiente código se genera el error VJS1213:

```
// VJS1213.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            case i:    // VJS1213
                // try the following line instead
                // case 0:
                default:
            }
        }
    }
}
```



# Error del compilador VJS1214

[Compiler Error VJS1214]

## La expresión switch debe ser un ordinal

El valor pasado a una instrucción **switch** no era un ordinal.

En el siguiente código se genera el error VJS1214:

```
// VJS1214.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch ("abc")    // VJS1214, string not an ordinal
        // try the following line instead
        // switch (i)
        {
            case 0:
            default:
        }
    }
}
```

# Error del compilador VJS1215

[Compiler Error VJS1215]

## Una expresión **synchronized** debe ser un tipo de referencia

El valor pasado a una expresión **synchronized** debe ser un tipo de referencia.

En el siguiente código se genera el error VJS1215:

```
// VJS1215.js1
class Test
{
    public static void main()
    {
        boolean tf = true;
        Test t = new Test();
        synchronized(tf)    // VJS1215, to resolve, pass t to synchronized
        {
            synchronized(tf)
            {
            }
        }
    }
}
```

# Error del compilador VJS1216

[Compiler Error VJS1216]

## Instrucción inalcanzable

El compilador encontró una instrucción que no se ejecutará nunca.

En el siguiente código se genera el error VJS1216:

```
// VJS1216.js1
class Test
{
    public static void main()
    {
        int i = 1;
        do
        {
            break;
            i++; // VJS1216, break statement makes this unreachable
        } while (i != 0);
    }
}
```

# Error del compilador VJS1217

[Compiler Error VJS1217]

**Esta instrucción no realiza ninguna acción**

El compilador encontró una instrucción que no tiene ningún efecto.

En el siguiente código se genera el error VJS1217:

```
// VJS1217.js1
class Test
{
    public static void main()
    {
        int i = 1;
        i;    // VJS1217, no effect
    }
}
```

# Error del compilador VJS1219

[Compiler Error VJS1219]

**Es necesaria una expresión en este contexto, no un nombre de tipo '*MyClass*'**

Se encontró un nombre de tipo donde se esperaba una expresión.

En el siguiente código se genera el error VJS1219:

```
// VJS1219.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (MyClass)    // VJS1219
        // try the following line instead
        // switch (i)
        {
            case 0:
            default:

        }
    }
}
```

# Error del compilador VJS1220

[Compiler Error VJS1220]

## El nombre '*nombre*' no se puede resolver

El compilador encontró una referencia a un miembro, pero el miembro al que se hace referencia no existe en este tipo.

En el siguiente código se genera el error VJS1220:

```
// VJS1220.js1
public class MyClass
{
    /*
    public static class InnerClass
    {
        public static void Test()
        {
        }
    }
    */
    public static void main (String [] args)
    {
        MyClass.InnerClass.Test();    // VJS1220, uncomment InnerClass to resolve
    }
}
```

# Error del compilador VJS1222

[Compiler Error VJS1222]

**La instrucción continue no se puede ejecutar fuera de un contexto de bucle**

Se encontró una instrucción **continue** fuera de un bucle.

En el siguiente código se genera el error VJS1222:

```
// VJS1222.js1
public class MyClass
{
    public static void main()
    {
        continue;    // VJS1222
    }
}
```

# Error del compilador VJS1223

[Compiler Error VJS1223]

**No se puede encontrar el método '*método*' en '*tipo*'**

La firma de un método no coincidió con una llamada al método.

En el siguiente código se genera el error VJS1223:

```
// VJS1223.js1
public class MyClass
{
    public void Test()
    {
    }

    public static void main()
    {
        MyClass x = new MyClass();
        x.Test(1);    // VJS1223
        // try the following line instead
        // x.Test();
    }
}
```



# Error del compilador VJS1224

[Compiler Error VJS1224]

**No se puede conceder el calificativo de súper a la llamada de la clase '*clase*'. La superclase no es una clase interna**

El compilador no pudo resolver una llamada a **super** porque la superclase no es una clase interna.

En el siguiente código se genera el error VJS1224:

```
// VJS1224.js1
public class MyClass
{
    public MyClass ()
    {
        new Class1 ().super ();    // VJS1224, delete this line to resolve
    }
    public static void main (String [] args)
    {
    }
}

public class Class1
{
}
```

# Error del compilador VJS1225

[Compiler Error VJS1225]

**Debe conceder el calificativo de súper a la llamada de la clase '*clase*'. La superclase es una clase interna**

Si un tipo extiende una clase interna no estática, no se puede llamar al constructor super directamente; se necesita una instancia de la clase envolvente de esa clase interna.

En el siguiente código se genera el error VJS1225:

```
// VJS1225.js1
public class MyClass extends Class1.Inner
{
    MyClass()
    {
        super();    // VJS1225
        // try the following line to resolve
        // new Class1(). super();
    }
    public static void main (String [] args)
    {
    }
}

public class Class1
{
    public class Inner
    // or, make the inner class static
    // public static class Inner
    {
    }
}
```

# Error del compilador VJS1227

[Compiler Error VJS1227]

**No se puede encontrar el constructor '*constructor*'**

La firma de un constructor no coincidió con una llamada al constructor.

En el siguiente código se genera el error VJS1227:

```
// VJS1227.js1
public class MyClass
{
    // MyClass(int i)
    // {
    // }
    public static void main()
    {
        MyClass x = new MyClass(9);    // VJS1227, uncomment c'tor above
        x.Test();
    }
    public void Test()
    {
    }
}
```

# Error del compilador VJS1228

[Compiler Error VJS1228]

**El método '*método*' ya está definido en la clase actual**

Se definió un método dos veces.

En el siguiente código se genera el error VJS1228:

```
// VJS1228.js1
public class MyClass
{
    public void Test()
    {
    }
    public static void main()
    {
    }
    public void Test()    // VJS1228, duplicate method definition
    {
    }
}
```

# Error del compilador VJS1231

[Compiler Error VJS1231]

## La llamada del método '*método*' es ambigua

El compilador no pudo determinar a qué método llamar.

En el siguiente código se genera el error VJS1231:

```
// VJS1231.js1

public class MyClass1
{
}

public class MyClass2 extends MyClass1
{
}

class MyClass3
{
    public void Test(MyClass2 mc2, MyClass1 mc1)
    {
    }
    public void Test(MyClass1 mc1, MyClass2 mc2)
    {
    }
    public static void main()
    {
        MyClass2 mc2 = new MyClass2();
        Test(mc2, mc2);    // VJS1231
    }
}
```

# Error del compilador VJS1232

[Compiler Error VJS1232]

**No puede haber una llamada explícita al método '*método*'**

El compilador produce algunos métodos internos en los que no se permite a los usuarios realizar llamadas explícitas.

# Error del compilador VJS1233

[Compiler Error VJS1233]

## Nombre de método incorrecto

Había un nombre de método que no era válido. Los caracteres válidos para un identificador son:

- Letras A-Z y a-z,
- Caracteres '\_' y '\$'
- Dígitos 0-9

En el siguiente código se genera el error VJS1233:

```
// VJS1233.jsl
public class MyClass
{
    public static void main (String [] args)
    {
        1234();    // VJS1233, invalid method name
    }
}
```

# Error del compilador VJS1234

[Compiler Error VJS1234]

**Una invocación explícita a 'this' o 'super' sólo se permite como primera instrucción de un constructor**

El uso de **this** y **super** está limitado a la primera línea cuando se utiliza en un constructor.

En el siguiente código se genera el error VJS1234:

```
// VJS1234.js1
public class MyClass1
{
    MyClass1()
    {
    }
}

public class MyClass2 extends MyClass1
{
    int i;
    MyClass2()
    {
        i = 0;
        super();    // VJS1234, make this the first line
    }
    public static void main()
    {
    }
}
```



# Error del compilador VJS1237

[Compiler Error VJS1237]

**La excepción '*excepción*' no se ha detectado y no aparece en la cláusula throws**

Una cláusula **throws** debe especificar las excepciones iniciadas desde un método o se debe detectar la excepción utilizando la cláusula **try-catch**.

En el siguiente código se genera el error VJS1237:

```
// VJS1237.js1
// compile with: /target:library
public class MyClass
{
    void method () // throws Exception
    {
        throw new Exception ();    // VJS1237
        // Exception should be caught or declared using throws clause
    }
}
```

# Error del compilador VJS1238

[Compiler Error VJS1238]

**La excepción '*excepción*' no puede iniciarla ni un inicializador estático ni un inicializador de campo estático**

Las excepciones activadas (no de tiempo de ejecución) no se pueden iniciar en los bloques del inicializador estático.

En el siguiente código se genera el error VJS1238:

```
// VJS1238.js1
class MyClass
{
    static
    {
        throw new java.io.IOException();    // VJS1238
        // OK to throw run-time exceptions
        // throw new NullPointerException();
    }
}
```

# Error del compilador VJS1239

[Compiler Error VJS1239]

**No se detecta la excepción '*excepción*' iniciada por el inicializador de campo o el constructor y no aparece en la cláusula *throws* del constructor**

Un constructor inició una excepción, pero ésta no estaba declarada en la declaración.

En el siguiente código se genera el error VJS1239:

Si un método inicia una excepción utilizando una instrucción **throw**, se debe declarar en la declaración del método utilizando una cláusula **throws**.

```
// VJS1239a.jsl
public class MyClass
{
    MyClass() // throws Exception
    {
        throw new Exception();    // VJS1239, uncomment throws clause in declaration
    }
    public static void main (String [] args) throws Exception
    {
        new MyClass ();
    }
}
```

Si un método inicia una excepción utilizando la instrucción **throws**, se debe detectar utilizando un bloque **try-catch**.

```
// VJS1239b.jsl
public class MyClass    // VJS1239, uncomment the following lines to resolve
{
    // {
    //     try
    //     {
    //         int x = Test();
    //     } catch (Exception e) {
    //     }
    // }
    int Test() throws Exception
    {
        return 10;
    }
    public static void main (String [] args)
    {
        new MyClass ();
    }
}
```

# Error del compilador VJS1240

[Compiler Error VJS1240]

## Un método de instancia no puede reemplazar al método estático '*método*'

Un método que reemplaza debe utilizar el mismo estado de instancia o estático que el método que se va a reemplazar.

En el siguiente código se genera el error VJS1240:

```
// VJS1240.js1
class MyClass1
{
    public static void Test()
    {
    }
}

class MyClass2 extends MyClass1
{
    public void Test()    // VJS1240, static/instance must agree with MyClass1 Test
    // try the following line instead
    // public static void Test()
    {
    }
    public static void main()
    {
        MyClass2 x = new MyClass2();
        x.Test();
    }
}
```

# Error del compilador VJS1241

[Compiler Error VJS1241]

## Un método estático no puede reemplazar al método de instancia '*método*'

Un método que reemplaza debe utilizar el mismo estado de instancia o estático que el método que se va a reemplazar.

En el siguiente código se genera el error VJS1241:

```
// VJS1241.js1
class MyClass1
{
    public void Test()
    {
    }
}

class MyClass2 extends MyClass1
{
    public static void Test()    // VJS1241, static/instance must agree with MyClass1 Test
    // try the following line instead
    // public void Test()

    {
    }
    public static void main()
    {
        MyClass2 x = new MyClass2();
        x.Test();
    }
}
```

# Error del compilador VJS1242

[Compiler Error VJS1242]

**No se puede reemplazar el método final '*método*'**

No se puede reemplazar el método final de una clase base.

En el siguiente código se genera el error VJS1242:

```
// VJS1242.js1
// compile with: /target:library
public class Base
{
    public static final int Test()
    {
        return 0;
    }
}

public class Derived extends Base
{
    public static int Test()    // VJS1242, hides Base.Test
    {
        return 1;
    }
}
```

# Error del compilador VJS1243

[Compiler Error VJS1243]

**No se puede reemplazar el método final '*método*'**

Un método **final** no se puede reemplazar.

En el siguiente código se genera el error VJS1243:

```
// VJS1243.js1
class MyClass1
{
    public final void Test()
    {
    }
}

class MyClass2 extends MyClass1
{
    public void Test()    // VJS1243, MyClass1.Test is final
    {
    }
    public static void main()
    {
        MyClass2 x = new MyClass2();
        x.Test();
    }
}
```

# Error del compilador VJS1245

[Compiler Error VJS1245]

**Los métodos '*método1*' y '*método2*' sólo se diferencian en los tipos devueltos**

Un método que reemplaza debe tener el mismo tipo de valor devuelto que el método al que reemplaza.

En el siguiente código se genera el error VJS1245:

```
// VJS1245.js1
class MyClass1
{
    public void Test()
    {
    }
}

class MyClass2 extends MyClass1
{
    public int Test()    // VJS1245, MyClass1.Test returns void
    {
    }
    public static void main()
    {
    }
}
```



# Error del compilador VJS1246

[Compiler Error VJS1246]

**El método '*método*' inicia '*excepción*' pero no en cláusulas throws de supermétodo**

Un método que reemplaza no tenía la misma especificación de excepción que el método al que reemplaza.

En el siguiente código se genera el error VJS1244:

```
// VJS1246.js1
public class MyClass extends Base
{
    public void Method() throws Exception    // VJS1246
    // uncomment exception specification in base method to resolve
    {
    }
    public static void main (String [] args)
    {
    }
}

class Base
{
    public void Method() // throws Exception
    {
    }
}
```

# Error del compilador VJS1247

[Compiler Error VJS1247]

## El método '*método*' define de nuevo el supermétodo con acceso más restrictivo

Un método que reemplaza no puede tener un acceso más restrictivo que el método al que reemplaza.

En el siguiente código se genera el error VJS1247:

```
// VJS1247.js1
public class MyClass1
{
    public void Text()
    {
    }
}

public class MyClass2 extends MyClass1
{
    private void Text()    // VJS1247, base class Text is public
    {
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1248

[Compiler Error VJS1248]

**No se puede obtener acceso al método '*método*'**

No había un método accesible.

En el siguiente código se genera el error VJS1248:

```
// VJS1248.js1
public class MyClass1
{
    private void Test()
    {
    }
}

class MyClass2 extends MyClass1
{
    public static void main()
    {
        MyClass1 x = new MyClass1();
        x.Test();    // VJS1248, Test is private
    }
}
```

# Error del compilador VJS1249

[Compiler Error VJS1249]

**No se puede tener acceso al miembro heredado '*miembro*' a través de este objeto**

No se puede tener acceso a *miembro* debido a su nivel de accesibilidad.

En el siguiente código se genera el error VJS1249:

```
// VJS1249a.jsl
// compile with: /target:library
package p;
public class MyClass
{
    protected void m(){ }
}
```

```
// VJS1249b.jsl
// compile with: VJS1249a.jsl
public class MyClass2 extends p.MyClass
{
    public static void main(String[] args)
    {
        new p.MyClass().m();    // VJS1249
    }
}
```

# Error del compilador VJS1250

[Compiler Error VJS1250]

**El prefijo del método '*método*' debe ser una referencia, no '*tipo*'**

Una llamada a un método debe estar calificada por una referencia al tipo que contiene el método.

En el siguiente código se genera el error VJS1250:

```
// VJS1250.js1
public class MyClass
{
    public static void main (String [] args)
    {
        MyClass x = new MyClass();
        int i;
        i = 10;
        i.Test();    // VJS1250
        // try the following line instead
        // x.Test();
    }
    public void Test()
    {
    }
}
```

# Error del compilador VJS1251

[Compiler Error VJS1251]

## Falta el tipo de valor devuelto para el método '*método*'

A un método le faltaba la especificación de tipo de valor devuelto.

En el siguiente código se genera el error VJS1251:

```
// VJS1251.js1
public class MyClass
{
    public static main()    // VJS1251
    // try the following line instead
    // public static void main()
    {
    }
}
```

# Error del compilador VJS1252

[Compiler Error VJS1252]

## Se debe poder asignar el parámetro real de '*parámetro*'

Se llamó a un método que toma un parámetro como referencia, pero el parámetro pasado no se puede asignar como literal; hay que pasar una variable.

En el siguiente código se genera el error VJS1252:

```
// VJS1252a.cs
// compile with: /target:library
// a C# program
public class RefClass
{
    public void method (ref int x)
    {
    }
}
```

```
// VJS1252b.jsl
// compile with: /reference:VJS1252a.dll
public class MyClass
{
    public static void main (String [] args)
    {
        int x = 10;
        new RefClass ().method (10);    // VJS1252
        // try the following line instead
        // new RefClass ().method (x);
    }
}
```

# Error del compilador VJS1253

[Compiler Error VJS1253]

**No se puede pasar el objeto de tipo '*tipo1*' como parámetro del tipo '*tipo2*'**

Se pasó un parámetro de referencia, pero el tipo no coincidía con el tipo declarado en el método.

En el siguiente código se genera el error VJS1253:

```
// VJS1253a.cs
// compile with: /target:library
// a C# program
public class RefClass
{
    public void Test(ref int x)
    {
    }
}
```

```
// VJS1253b.jsl
// compile with: /reference:VJS1253a.dll
class MyClass
{
    public static void main (String [] args)
    {
        short x = 0;
        new RefClass ().Test(x);    // VJS1253

        int y = 0;
        new RefClass ().Test(y);    // OK
    }
}
```



# Error del compilador VJS1254

[Compiler Error VJS1254]

## El método '*método*' definido en una interfaz no puede tener cuerpo

Los métodos de una interfaz se pueden declarar, pero no definir.

En el siguiente código se genera el error VJS1254:

```
// VJS1254.js1
interface MyInterface
{
    void Test() {}    // VJS1254
    // try the following line instead
    // void Test();
}

public class MyClass1
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1255

[Compiler Error VJS1255]

## El método abstracto '*método*' no puede tener cuerpo

Los métodos marcados como **abstract** se pueden declarar, pero no definir.

En el siguiente código se genera el error VJS1255:

```
// VJS1255.js1
abstract public class MyClass
{
    public abstract void Test(){}    // VJS1255
    // try the following line instead
    // public abstract void Test();
    // or delete the abstract keyword and keep the method body
    public static void main()
    {
    }
}
```

# Error del compilador VJS1256

[Compiler Error VJS1256]

## El método nativo '*método*' no puede tener cuerpo

Los métodos marcados como **native** se pueden declarar, pero no definir.

En el siguiente código se genera el error VJS1256:

```
// VJS1256.js1
//
public class MyClass
{
    public native void Test(){}    // VJS1256
    // try the following line instead
    // public native void Test();
    // or delete the native keyword and keep the method body
    public static void main()
    {
    }
}
```

# Error del compilador VJS1257

[Compiler Error VJS1257]

**El método '*método*' no es abstracto y no tiene cuerpo**

No se definió un método en una clase.

En el siguiente código se genera el error VJS1257:

```
// VJS1257.js1
//
public class MyClass
{
    public void Test();    // VJS1257
    // try the following line instead
    // public void Test(){}
    public static void main()
    {
    }
}
```

# Error del compilador VJS1259

[Compiler Error VJS1259]

## El método '*método*' debe devolver un valor

La firma de un método especificó un valor devuelto, pero no se encontró una instrucción return en el cuerpo de la implementación del método.

En el siguiente código se genera el error VJS1259:

```
// VJS1259.js1
public class MyClass
{
    public int Test()    // VJS1259, uncomment return statement to resolve
    {
        // return 0;
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1265

[Compiler Error VJS1265]

## La instrucción return no está permitida en un inicializador

El compilador encontró una instrucción return en un bloque de inicializador.

En el siguiente código se genera el error VJS1265:

```
// VJS1265.js1
public class MyClass    // VJS1265
{
    {
        return;    // return not allowed in initializer block
    }
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1271

[Compiler Error VJS1271]

## Sólo se permite un modificador de acceso

Había un modificador de acceso repetido.

En el siguiente código se genera el error VJS1271:

```
// VJS1271.js1
public class MyClass
{
    public private void Test()    // VJS1271, delete one public
    {
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1274

[Compiler Error VJS1274]

## No se pueden crear atributos, enumeraciones ni tipos de valor

Los tipos de valor, las enumeraciones y los atributos no se pueden extender.

En el siguiente código se genera el error VJS1274:

```
// VJS1274a.jsl
class Class1 extends System.ValueType    // VJS1274
{
}
```

```
// VJS1274b.jsl
class Class2 extends System.Enum        // VJS1274
{
}
```

```
// VJS1274c.jsl
class Class3 extends System.Attribute    // VJS1274
{
}
```



# Error del compilador VJS1275

[Compiler Error VJS1275]

**El nombre de clase '*nombre*' está en conflicto con un nombre de paquete**

El nombre de una clase en un módulo era el mismo que el de un paquete en otro módulo.

En el siguiente código se genera el error VJS1275:

```
// VJS1275a.jsl
// compile with: /target:library
package Test;
public class MyClass
{
}
```

```
// VJS1275b.jsl
// compile with: VJS1275a.jsl
public class Test    // VJS1275
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1276

[Compiler Error VJS1276]

**La etiqueta '*etiqueta*' utilizada en la instrucción continue debe señalar a una instrucción loop**

Una etiqueta en una instrucción **continue** no señalaba un bucle.

En el siguiente código se genera el error VJS1276:

```
// VJS1276.js1
public class MyClass
{
    public static void main()
    {
        label1: for (int i = 0; i < 100; ++i)
        {
            label2:
            {
                continue label2;    // VJS1276
                // try the following line instead
                // continue label1;
            }
        }
    }
}
```

# Error del compilador VJS1277

[Compiler Error VJS1277]

**'miembro' no es compatible**

No se puede tener acceso a un miembro que no es compatible.

En el siguiente código se genera el error VJS1277:

```
// VJS1277a.cs
// compile with: /target:library
// a C# program
public class MyClass
{
    [System.ObsoleteAttribute("", true)]
    public static void m() { }
}
```

```
// VJS1277.jsl
// compile with: /reference:vjs1277a.dll
public class MyClass2
{
    public static void main(String[] args)
    {
        MyClass.m();    // VJS1277
    }
}
```

# Error del compilador VJS1278

[Compiler Error VJS1278]

**Los métodos '*método1*' y '*método2*' no se pueden definir en la misma clase**

El siguiente conjunto de métodos no se puede utilizar en la misma clase:

- `String toString()` y `String ToString()`
- `Boolean equals(Object)` y `Boolean Equals(Object)`

En el siguiente código se genera el error VJS1278:

```
// VJS1278.jsl
public class MyClass    // VJS1278
{
    public String toString() {}
    public String ToString() {}
    public static void main(String[] args)
    {
        new MyClass() { };
    }
}
```

# Error del compilador VJS1279

[Compiler Error VJS1279]

'*miembro*' debe implementarlo la clase anónima

Una clase anónima no es nunca abstracta y debe implementar todos los métodos abstractos que contiene.

En el siguiente código se genera el error VJS1279:

```
// VJS1279.js1
public abstract class MyClass
{
    abstract void m();
    public static void main(String[] args)
    {
        new MyClass() { };    // VJS1279
    }
}
```

# Error del compilador VJS1280

[Compiler Error VJS1280]

**'*tipo*' no es una clase anidada ni una interfaz de la clase '*clase*'**

El compilador detectó un intento de crear una instancia de un tipo que no es una clase anidada ni una interfaz de la clase.

En el siguiente código se genera el error VJS1280:

```
// VJS1280.js1
public class MyClass
{
    public static void main(String[] args)
    {
        new MyClass().new Inner();    // VJS1280
    }
}
```

# Error del compilador VJS1281

[Compiler Error VJS1281]

**Miembro ambiguo: '*member1*' heredado y '*miembro2*' de ámbito externo. Se requiere un calificador explícito 'this'**

El compilador no puede determinar el miembro al que desea tener acceso.

En el siguiente código se genera el error VJS1281:

```
// VJS1281.js1
// compile with: /target:library
class Base {
    int i;
}

class Outer {
    int i;
    class Derived extends Base {
        int j = i;
    }
}
```

# Error del compilador VJS1282

[Compiler Error VJS1282]

## No se puede declarar el tipo '*type*'

Los tipos **System.Object** y **System.String** no se pueden declarar en un archivo de código fuente.

En el siguiente código se genera el error VJS1282:

```
// VJS1282.js1
// compile with: /target:library /nowarn:1513
package System;    // VJS1282
class Object {
}
```



# Error del compilador VJS1300

[Compiler Error VJS1300]

**'*valor*' no está en el intervalo de '*tipo*'**

Se asignó un valor a un tipo, pero el valor está fuera del intervalo de los compatibles para ese tipo.

En el siguiente código se genera el error VJS1301:

```
// VJS1300.js1
class MyClass
{
    public static void main (String [] args)
    {
        byte i = 256;    // VJS1300, 256 out of range for byte
        // try the following line instead
        // byte i = 127;
    }
}
```

# Error del compilador VJS1301

[Compiler Error VJS1301]

**El tipo '*tipo1*' no se puede asignar a '*tipo2*'**

Se intentó una asignación no válida.

En el siguiente código se genera el error VJS1301:

```
// VJS1301.js1
public class MyClass
{
    public static void main()
    {
        int i = 2.2;    // VJS1301
        // try the following lines instead
        // int i = 2;
        // i++;
    }
}
```

# Error del compilador VJS1302

[Compiler Error VJS1302]

**No se puede tener acceso al campo no estático '*campo*' desde un contexto estático**

Hubo un intento de acceso a un campo de instancia como si fuese un campo estático.

En el siguiente código se genera el error VJS1302:

```
// VJS1302.js1
// compile with: /target:library
class MyClass
{
    public void MyMethod()
    {
        Test.x = 10;    // VJS1302 x is not static in class Test
        int local = new Test ().x;    // OK, accessing instance field
        Test.y = 20;    // OK, accessing static field
    }
}

class Test
{
    public int x;    // instance field
    public static int y;    // static field
}
```

# Error del compilador VJS1303

[Compiler Error VJS1303]

**No se puede tener acceso al método no estático '*método*' desde un contexto estático**

A un método de instancia se debe tener acceso por medio de una instancia de su tipo envolvente.

En el siguiente código se genera el error VJS1303:

```
// VJS1303.js1
class MyClass
{
    public void Test()
    {
    }
    public static void main (String [] args)
    {
        MyClass.Test();    // VJS1303
        // try the following lines instead
        // MyClass x = new MyClass();
        // x.Test();
    }
}
```

# Error del compilador VJS1304

[Compiler Error VJS1304]

**No se puede tener acceso al constructor de la clase '*clase*'**

No se pudo tener acceso a un constructor.

En el siguiente código se genera el error VJS1304:

```
// VJS1304.js1
class MyClass
{
    public static void main (String [] args)
    {
        new Class1();    // VJS1304  constructor is private
        new Class1(10);  // OK, constructor is public
    }
}

class Class1
{
    private Class1()
    {
    }
    public Class1(int x)
    {
    }
}
```

# Error del compilador VJS1305

[Compiler Error VJS1305]

## Una condición en una expresión condicional debe ser booleana

La expresión evaluada en una expresión condicional debe ser una expresión booleana.

En el siguiente código se genera el error VJS1305:

```
// VJS1305.js1
class Test
{
    public static void main(String[] args)
    {
        int i = 0;
        boolean j = (i) ? true : false;    // VJS1305
        // try the following line instead
        // boolean j = (i == 0) ? true : false;
    }
}
```

# Error del compilador VJS1306

[Compiler Error VJS1306]

**Los tipos '*tipo1*' y '*tipo2*' son incompatibles con una expresión condicional**

Había una expresión condicional que contenía tipos incompatibles.

En el siguiente código se genera el error VJS1306:

```
// VJS1306.js1
class MyClass
{
    public static void main (String [] args)
    {
        int j = 3 < 10 ? 1 : new Object();    // VJS1306 object not int
        int k = 3 < 10 ? 1 : 3;    // OK
    }
}
```

# Error del compilador VJS1307

[Compiler Error VJS1307]

**No se puede crear un objeto nuevo de tipo '*tipo*' porque es una clase abstracta**

No se pueden crear instancias de una clase abstracta.

En el siguiente código se genera el error VJS1307:

```
// VJS1307.js1
abstract public class MyClass1
{
    public abstract void Test();
}

public class MyClass2
{
    public static void main()
    {
        MyClass1 x = new MyClass1();    // VJS1307
    }
}
```



# Error del compilador VJS1308

[Compiler Error VJS1308]

**No se puede crear un objeto nuevo de tipo '*tipo*' porque es una interfaz**

No se puede crear una instancia de un objeto desde una interfaz.

En el siguiente código se genera el error VJS1308:

```
// VJS1308.js1
class MyClass
{
    public static void main (String [] args)
    {
        MyInterface i = new MyInterface();    // VJS1308
    }
}

interface MyInterface
{
}
```

# Error del compilador VJS1309

[Compiler Error VJS1309]

**No se puede crear una instancia de un tipo primitivo con 'new'**

No se puede utilizar el operador **new** para crear una instancia de un tipo primitivo.

En el siguiente código se genera el error VJS1309:

```
// VJS1309.js1
class Test
{
    public static void main(String[] args)
    {
        int i = new int();    // VJS1309
        // try the following line instead
        // int i;
    }
}
```

# Error del compilador VJS1312

[Compiler Error VJS1312]

## No se puede aplicar un operador postfijo a '*tipo*'

Se aplicó el operador postfijo de incremento (++) a una variable cuyo tipo no admite este operador.

En el siguiente código se genera el error VJS1312:

```
// VJS1312.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        x++;    // VJS1312
    }
}
```

# Error del compilador VJS1313

[Compiler Error VJS1313]

**No se puede aplicar un operador postfijo a un elemento que no se puede asignar**

Se aplicó el operador postfijo a un símbolo que no se podía llevar a cabo.

En el siguiente código se genera el error VJS1313:

```
// VJS1313.js1
class Test
{
    public static void main(String[] args)
    {
        int i = 0;
        0++;    // VJS1313
        // try the following line instead
        // i++;
    }
}
```

# Error del compilador VJS1314

[Compiler Error VJS1314]

## No se puede aplicar un operador postfijo a una variable no inicializada

El operador postfijo de incremento (++) se aplicó a una variable que no se había podido inicializar aún.

En el siguiente código se genera el error VJS1314:

```
// VJS1314.js1
class Test
{
    public static void main(String[] args)
    {
        int i;
        i++;    // VJS1314, initialize i first
    }
}
```

# Error del compilador VJS1315

[Compiler Error VJS1315]

**No se puede aplicar un operador prefijo a '*tipo*'**

Se aplicó el operador prefijo de incremento (++) a una variable cuyo tipo no admite este operador.

En el siguiente código se genera el error VJS1315:

```
// VJS1315.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        --x;    // VJS1315
    }
}
```

# Error del compilador VJS1316

[Compiler Error VJS1316]

**No se puede aplicar un operador prefijo a un elemento que no se puede asignar**

Se aplicó el operador prefijo a un símbolo que no se podía llevar a cabo.

En el siguiente código se genera el error VJS1316:

```
// VJS1316.js1
class Test
{
    public static void main(String[] args)
    {
        int i = 0;
        --0;    // VJS1316
        // try the following line instead
        // --i;
    }
}
```

# Error del compilador VJS1317

[Compiler Error VJS1317]

## No se puede aplicar un operador prefijo a una variable no inicializada

El operador prefijo de incremento (++) se aplicó a una variable que no se había podido inicializar aún.

En el siguiente código se genera el error VJS1317:

```
// VJS1317.js1
class Test
{
    public static void main(String[] args)
    {
        int i;
        --i;    // VJS1317, initialize i first
    }
}
```



# Error del compilador VJS1318

[Compiler Error VJS1318]

**No se puede aplicar un operador unario +/- a '*tipo*'**

Los operadores unarios se pueden utilizar únicamente con tipos numéricos.

En el siguiente código se genera el error VJS1318:

```
// VJS1318.js1
class MyClass
{
    public static void main (String [] args)
    {
        Object obj =- (new Object());    // VJS1318
    }
}
```

# Error del compilador VJS1319

[Compiler Error VJS1319]

**No se puede aplicar un operador unario ~ a '*operando*'**

El operador unario ~ no es válido en un operando.

En el siguiente código se genera el error VJS1319:

```
// VJS1319.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        x = ~x;    // VJS1319
        // try the following lines instead
        // byte i = 25;
        // byte j = ~25;
    }
}
```

# Error del compilador VJS1320

[Compiler Error VJS1320]

**No se puede aplicar un operador unario ! a '*tipo*'**

Se aplicó el operador unario ! a un tipo que no admite negación.

En el siguiente código se genera el error VJS1320:

```
// VJS1320.js1
class Test
{
    public static void main(String[] args)
    {
        int i;
        boolean j = true;
        i = 0;
        if (!i)    // VJS1320
        // try the following line instead
        // if (!j)
        {
        }
    }
}
```

# Error del compilador VJS1321

[Compiler Error VJS1321]

**No se puede aplicar el operador "==" a '*tipo1*' y a '*tipo2*'**

Se intentó una comparación entre tipos que no admiten comparaciones.

En el siguiente código se genera el error VJS1321:

```
// VJS1321.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x == 1)    // VJS1321
        {
        }
    }
}
```

# Error del compilador VJS1322

[Compiler Error VJS1322]

**No se puede aplicar un operador relacional a '*tipo1*' y a '*tipo2*'**

Se intentó una operación relacional entre tipos que no admiten operaciones relacionales.

En el siguiente código se genera el error VJS1322:

```
// VJS1322.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x < 1)    // VJS1322
        {
        }
    }
}
```

# Error del compilador VJS1323

[Compiler Error VJS1323]

**No se puede aplicar el operador "+" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación de suma entre tipos que no admiten operaciones de suma.

En el siguiente código se genera el error VJS1323:

```
// VJS1323.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x + 1)    // VJS1323
        {
        }
    }
}
```

# Error del compilador VJS1324

[Compiler Error VJS1324]

**No se puede aplicar el operador "\*" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación de multiplicación entre tipos que no admiten operaciones de este tipo.

En el siguiente código se genera el error VJS1324:

```
// VJS1324.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x * 1)    // VJS1324
        {
        }
    }
}
```

# Error del compilador VJS1325

[Compiler Error VJS1325]

**No se puede aplicar el operador "/" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación de división entre tipos que no admiten operaciones de este tipo.

En el siguiente código se genera el error VJS1325:

```
// VJS1325.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x / 1)    // VJS1325
        {
        }
    }
}
```



# Error del compilador VJS1326

[Compiler Error VJS1326]

**No se puede aplicar el operador "-" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación de resta entre tipos que no admiten operaciones de resta.

En el siguiente código se genera el error VJS1326:

```
// VJS1326.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x - 1)    // VJS1326
        {
        }
    }
}
```

# Error del compilador VJS1327

[Compiler Error VJS1327]

**No se puede aplicar el operador "%" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación de módulo entre tipos que no admiten operaciones de este tipo.

En el siguiente código se genera el error VJS1327:

```
// VJS1327.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x % 1)    // VJS1327
        {
        }
    }
}
```

# Error del compilador VJS1328

[Compiler Error VJS1328]

**No se pueden aplicar los operadores bit a bit &, ^, | a '*tipo1*' y a '*tipo2*'**

Se intentó una operación entre tipos que no admiten dicha operación.

En el siguiente código se genera el error VJS1328:

```
// VJS1328.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x & 1)    // VJS1328
        {
        }
    }
}
```

# Error del compilador VJS1329

[Compiler Error VJS1329]

**No se pueden aplicar los operadores de desplazamiento <<, >>, >>> a '*tipo1*' y a '*tipo2*'**

Se intentó una operación entre tipos que no admiten dicha operación.

En el siguiente código se genera el error VJS1329:

```
// VJS1329.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x << 1)    // VJS1329
        {
        }
    }
}
```

# Error del compilador VJS1330

[Compiler Error VJS1330]

## No se puede dividir un número por cero

La división por cero no está definida y, por tanto, no es una operación válida.

En el siguiente código se genera el error VJS1330:

```
// VJS1330.js1
class Test
{
    public static void main(String[] args)
    {
        int i = 1/0;    // VJS1330
    }
}
```

# Error del compilador VJS1331

[Compiler Error VJS1331]

**No se puede aplicar el operador "&&" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación entre tipos que no admiten dicha operación.

En el siguiente código se genera el error VJS1331:

```
// VJS1331.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x && 1)    // VJS1331
        {
        }
    }
}
```

# Error del compilador VJS1332

[Compiler Error VJS1332]

**No se puede aplicar el operador "||" a '*tipo1*' y a '*tipo2*'**

Se intentó una operación entre tipos que no admiten dicha operación.

En el siguiente código se genera el error VJS1332:

```
// VJS1332.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x || 1)    // VJS1332
        {
        }
    }
}
```

# Error del compilador VJS1333

[Compiler Error VJS1333]

**No se puede realizar la conversión de '*tipo1*' y '*tipo2*'**

Se intentó una conversión entre tipos que no admiten conversión.

En el siguiente código se genera el error VJS1333:

```
// VJS1333.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        int i = 0;
        i = (int)x;    // VJS1333
    }
}
```



# Error del compilador VJS1334

[Compiler Error VJS1334]

**El tipo '*tipo1*' no se puede devolver como '*tipo2*'**

Un método devolvió un tipo diferente al esperado.

En el siguiente código se genera el error VJS1334:

```
// VJS1334.js1
class Test
{
    public int Test()
    {
        Test x = new Test();
        return x;    // VJS1334
        // try the following line instead
        // return 0;
    }
    public static void main(String[] args)
    {
    }
}
```

# Error del compilador VJS1335

[Compiler Error VJS1335]

## Debe devolver un valor de un método que no sea void

Un método tenía un tipo de valor devuelto distinto de void, pero el cuerpo del método tenía una instrucción return sin ningún valor.

En el siguiente código se genera el error VJS1335:

```
// VJS1335.js1
// compile with: /target:library
public class MyClass
{
    public int Test()
    {
        return;    // VJS1335
        // try the following line instead
        // return 0;
    }
}
```

# Error del compilador VJS1336

[Compiler Error VJS1336]

## No se puede devolver un valor desde un método void

Un método marcado como que tiene un tipo de valor devuelto void no puede devolver ningún valor.

En el siguiente código se genera el error VJS1336:

```
// VJS1336.js1
public class MyClass
{
    public static void Test()
    {
        return 0;    // VJS1336, delete return value
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1337

[Compiler Error VJS1337]

## No se puede devolver un valor desde un constructor

Un constructor no puede contener una instrucción return.

En el siguiente código se genera el error VJS1337:

```
// VJS1337.js1
class Test
{
    Test()
    {
        return 0;    // VJS1337, delete return statement
    }
    public static void main(String[] args)
    {
    }
}
```

# Error del compilador VJS1338

[Compiler Error VJS1338]

**No se puede aplicar el operador "instanceof" a '*tipo1*' y a '*tipo2*'**

Una operación **instanceof** relacionaba dos tipos incompatibles.

En el siguiente código se genera el error VJS1338:

```
// VJS1338.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x instanceof Test2)    // VJS1338
            // try the following line instead
            // if (x instanceof Test)
            {
            }
    }
}

class Test2
{
}
```

# Error del compilador VJS1339

[Compiler Error VJS1339]

**El lado derecho de 'instanceof' debe ser un tipo de referencia, no '*tipo*'**

El identificador situado a la derecha del operador **instanceof** debe ser un tipo de referencia.

En el siguiente código se genera el error VJS1339:

```
// VJS1339.js1
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x instanceof int)    // VJS1339
        // try the following line instead
        // if (x instanceof Test)
        {
        }
    }
}
```

# Error del compilador VJS1340

[Compiler Error VJS1340]

## Falta índice de matriz

En una operación de matriz, falta un índice en la matriz.

En el siguiente código se genera el error VJS1340:

```
// VJS1340.js1
class Test
{
    public static void main(String[] args)
    {
        int[] x = new int[5];
        x[] = 0;    // VJS1340
        // try the following line instead
        // x[0] = 0;
    }
}
```

# Error del compilador VJS1341

[Compiler Error VJS1341]

## No se pueden especificar los límites de la matriz con el inicializador

Cuando se inicializa una matriz utilizando una lista de inicialización, no se puede especificar la dimensión de la matriz.

En el siguiente código se genera el error VJS1341:

```
// VJS1341.js1
// compile with: /target:library
class MyClass
{
    public void Test()
    {
        int arr[] = new int[2]{1,2}; // VJS1341
        // try the following line instead
        // int arr[] = new int[] {1,2};
        arr[0] = 0;
    }
}
```



# Error del compilador VJS1342

[Compiler Error VJS1342]

**El descriptor de acceso de la matriz debe ser ordinal pero era '*tipo*'**

Al tener acceso a los elementos de una matriz, se utilizó un índice cuyo tipo no era un tipo integral.

En el siguiente código se genera el error VJS1342:

```
// VJS1342.js1
public class MyClass
{
    public static void main()
    {
        int arr[] = {1,2};
        String str = "test";
        arr[str] = 1;    // VJS1342, str is type String, not int
        // try the following line instead
        arr[0] = 1;
    }
}
```

# Error del compilador VJS1343

[Compiler Error VJS1343]

**El prefijo del descriptor de acceso de la matriz debe ser un tipo array pero era '*tipo*'**

Se utilizó el operador [] del descriptor de acceso de la matriz en una variable que no era una matriz.

En el siguiente código se genera el error VJS1343:

```
// VJS1343.js1
public class MyClass
{
    public static void main()
    {
        int i = 0;
        int arr[] = {1,2};
        i[0] = 1;    // VJS1343 i is not an array so i[0] is not valid
        // try the following line instead
        arr[0] = 1;
    }
}
```

# Error del compilador VJS1344

[Compiler Error VJS1344]

## El límite de la matriz debe ser ordinal pero era '*tipo*'

Cuando se especifica el tamaño de la matriz en, por ejemplo, una llamada al operador **new**, el tipo de la variable debe ser un ordinal.

En el siguiente código se genera el error VJS1344:

```
// VJS1344.js1
public class MyClass
{
    public static void main()
    {
        String s = "test";
        int i = 10;
        int arr[] = new int[s]; // VJS1344, s is String not ordinal
        // try the following line instead
        int arr2[] = new int[i];
    }
}
```

# Error del compilador VJS1345

[Compiler Error VJS1345]

**Se esperaba un valor de tipo '*tipo*', pero se encontró un inicializador de matriz**

Hubo un intento de inicializar un tipo que no era de matriz con un inicializador de matriz.

En el siguiente código se genera el error VJS1345:

```
// VJS1345.js1
public class MyClass
{
    int i;
    public static void main()
    {
        MyClass c = {1,2};    // VJS1345
        // try the following line instead
        MyClass c1 = new MyClass();
        c1.i = 0;
    }
}
```

# Error del compilador VJS1346

[Compiler Error VJS1346]

## La creación de una matriz debe especificar al menos una dimensión

Había una declaración de matriz mal formada porque faltaba la dimensión de la misma.

En el siguiente código se genera el error VJS1346:

```
// VJS1346.js1
class Test
{
    public static void main(String[] args)
    {
        int[] x = new int[];    // VJS1346
        // try the following line instead
        // int[] x = new int[5];
    }
}
```

# Error del compilador VJS1347

[Compiler Error VJS1347]

**Se deben especificar todos los límites de una matriz rectangular si se especifica alguno**

Había una declaración de matriz rectangular mal formada.

En el siguiente código se genera el error VJS1347:

```
// VJS1347.js1
class MyClass
{
    public static void main()
    {
        int[,] arr = new int[10,];    // VJS1347, a dimension is missing.
        int[,] arr2 = new int[10,10]; // OK, all dimensions are present
    }
}
```

# Error del compilador VJS1348

[Compiler Error VJS1348]

**No se permiten las dimensiones especificadas después de las no especificadas en la creación de una matriz**

No se especificó la primera dimensión de una matriz pero la segunda sí, y esto no está permitido.

En el siguiente código se genera el error VJS1348:

```
// VJS1348.js1
public class MyClass
{
    public static void main()
    {
        int[][] arr = new int[][2];    // VJS1348
        // try the following line instead
        int[][] arr2 = new int[3][2];
    }
}
```

# Error del compilador VJS1350

[Compiler Error VJS1350]

## Inicializador de matriz incorrecto para '*declaración*'

Se especificó un inicializador para una matriz rectangular, pero el inicializador no corresponde a una matriz rectangular.

En el siguiente código se genera el error VJS1350:

```
// VJS1350.js1
class MyClass
{
    public static void main()
    {
        int[,] arr = new int[,]{{1,2},{3,4,5}};    // VJS1350
        // Rectangular arrays should have all rows of the same size
        // try the following line instead
        int[,] arr2 = new int[,]{{1,2},{3,4}};
        arr2[0,0] = 0;
    }
}
```



# Error del compilador VJS1351

[Compiler Error VJS1351]

## Número erróneo de índices para '*declaración*'

Al intentar el acceso a los elementos de una matriz rectangular, el número de índices dado no coincidía con las dimensiones de la matriz.

En el siguiente código se genera el error VJS1351:

```
// VJS1351.js1
class MyClass
{
    public static void main()
    {
        int[,] arr = new int[2,2];    // 2x2 rectangular array
        arr[0] = 1;    // VJS1351, must specify two indices
        // try the following line instead
        arr[0,1] = 10;
    }
}
```

# Error del compilador VJS1354

[Compiler Error VJS1354]

## 'this' no está disponible en un contexto estático

El operador **this** no se puede utilizar en un método estático.

En el siguiente código se genera el error VJS1354:

```
// VJS1354.js1
class Test
{
    public static void Test(Test t)
    {
        if (this == t)    // VJS1354
        {
        }
    }
    public static void main(String[] args)
    {
    }
}
```

# Error del compilador VJS1355

[Compiler Error VJS1355]

## 'this' no está disponible en la llamada del constructor en el contexto actual

Al invocar a un constructor de clase utilizando la sintaxis **this**, no se pueden utilizar las variables de instancia de una clase. Estas variables no están disponibles, puesto que la variable **this** no se ha inicializado aún.

En el siguiente código se genera el error VJS1355:

```
// VJS1355.js1
// compile with: /target:library
public class MyClass
{
    int i;    // an instance variable
    static final int ONE = 1;    // a static variable
    public MyClass()
    {
        this(i);    // VJS1355
        // try the following line instead
        // this(ONE); // OK. ONE is a static variable
    }
    public MyClass(int i)
    {
    }
}
```

# Error del compilador VJS1356

[Compiler Error VJS1356]

**El prefijo para '.this' debe ser un nombre de clase, no '*tipo*'**

Había una llamada a **this** mal formada.

En el siguiente código se genera el error VJS1356:

```
// VJS1356.js1
// compile with: /target:library
public class OuterClass
{
    public int i;
    public class InnerClass
    {
        void Test()
        {
            int.this.i = 20;    // VJS1356, prefix int is not a class name
            // try the following line instead
            // OuterClass.this.i = 20; // OK
        }
    }
}
```

# Error del compilador VJS1357

[Compiler Error VJS1357]

## La clase '*clase*' no es una clase envolvente

Una instrucción **this** cualificada intentó tener acceso a la variable **this** de una clase envolvente; sin embargo, el prefijo de la expresión **this** no es una clase envolvente externa.

En el siguiente código se genera el error VJS1357:

```
// VJS1357.js1
// compile with: /target:library
public class OuterClass
{
    public int i;
    public class InnerClass
    {
        void Test()
        {
            InnerClass2.this.j = 10;    // VJS1357, not an enclosing class
            // try the following line instead
            // OuterClass.this.i = 10;
        }
    }

    public class InnerClass2
    {
        public int j;
    }
}
```

# Error del compilador VJS1358

[Compiler Error VJS1358]

## Se esperaba un bloque catch o finally

El compilador detectó una instrucción **try** incompleta.

En el siguiente código se genera el error VJS1358:

```
// VJS1358.js1
class MyException extends Exception
{
}

class MyClass
{
    public static void main(String[] args)
    {
        try    // VJS1358, uncomment the catch block to resolve
        {
        }
        // catch (Exception e)
        // {
        // }
    }
}
```

# Error del compilador VJS1359

[Compiler Error VJS1359]

**No se puede detectar '*clase*' que no se hereda de `java.lang.Throwable` o `System.Exception`**

Una excepción especificada en una cláusula **catch** no se derivaba de una clase de excepción.

En el siguiente código se genera el error VJS1359:

```
// VJS1359.js1
class MyClass
{
}

class MyClass2
{
    public static void main(String[] args)
    {
        try
        {
        }
        catch (MyClass e)    // VJS1359, MyClass is not an exception class
        // try the following line instead
        // catch (Exception e)
        {
        }
    }
}
```

# Error del compilador VJS1360

[Compiler Error VJS1360]

## Catch inaccesible: no se iniciaron excepciones correspondientes en el bloque try

No se puede alcanzar una cláusula **catch** porque no se inicia la clase de excepción coincidente o la excepción derivada en el bloque try.

En el siguiente código se genera el error VJS1360:

```
// VJS1360.js1
class MyClass extends Exception
{
}

class MyClass2
{
    public static void main(String[] args)
    {
        try
        {
            // uncomment the following line to resolve:
            //    throw new MyClass();
        }
        catch (MyClass e)    // VJS1360, Exception MyClass is not thrown in the try block
        {
        }
        catch (Exception e)
        {
        }
    }
}
```



# Error del compilador VJS1361

[Compiler Error VJS1361]

**Catch inaccesible: se detectaron excepciones coincidentes en una cláusula catch anterior**

El compilador detectó una cláusula **catch** duplicada.

En el siguiente código se genera el error VJS1361:

```
// VJS1361.js1
class MyClass
{
}

class MyClass2
{
    public static void main(String[] args)
    {
        try
        {
        }
        catch (Exception e)
        {
        }

        catch (Exception e)    // VJS1361, duplicate catch block
        {
        }
    }
}
```

# Error del compilador VJS1362

[Compiler Error VJS1362]

**No se puede iniciar '*clase*' que no se hereda de `java.lang.Throwable` o `System.Exception`**

Una instrucción **throw** inició una clase que no era una clase de excepción.

En el siguiente código se genera el error VJS1362:

```
// VJS1362.js1
class MyClass
{
}

class MyClass2 extends Exception
{
}

class MyClass3
{
    public static void main(String[] args)
    {
        try
        {
            throw new MyClass();    // VJS1362, MyClass is not exception class
            // try the following line instead
            // throw new MyClass2();
        }

        catch (MyClass2 e)
        {
        }
    }
}
```

# Error del compilador VJS1363

[Compiler Error VJS1363]

## No se puede obtener acceso a la clase '*clase*'

La clase (**class**) es de ámbito de paquete y una clase de otro módulo está intentando extenderla en un paquete diferente. Sólo se puede tener acceso a un miembro de ámbito de paquete en otra clase si la clase que tiene acceso está en el mismo paquete.

En el siguiente código se genera el error VJS1363:

```
// VJS1363a.jsl
// compile with: /target:library
package p;
class Base
// one solution is to make Base a public class
{
}
```

```
// VJS1363b.jsl
// compile with: /target:library VJS1363a.jsl
class MyClass extends p.Base // VJS1363
// one solution is to add MyClass to package p
{
}
```

# Error del compilador VJS1364

[Compiler Error VJS1364]

## El prefijo del campo '*campo*' debe ser una referencia, no '*tipo*'

El acceso a un campo de instancia se debe llevar a cabo por medio de una instancia del tipo.

Este error indica que intentó tener acceso a campos de un tipo primitivo, como **int**, **char** o **boolean**. Los tipos primitivos no pueden tener campos. Sólo los tipos de referencia pueden tener campos y métodos.

En el siguiente código se genera el error VJS1364:

```
// VJS1364.js1
class MyClass
{
    int i;
    public void Test()
    {
        MyClass x = new MyClass();
        i.i = 0;    // VJS1364, i is not a reference to the class
        // try the following line instead
        // x.i = 0;
    }
    public static void main()
    {
    }
}
```

# Error del compilador VJS1365

[Compiler Error VJS1365]

**No se puede encontrar el campo '*campo*' en la clase '*clase*'**

El compilador encontró una referencia a un campo que no estaba declarado en la clase.

En el siguiente código se genera el error VJS1365:

```
// VJS1365.jsl
class Test
{
    int i;
    public void Test(Test t)
    {
        this.t = 0;    // VJS1365
        // try the following line instead
        // this.i = 0;
    }
    public static void main(String[] args)
    {
    }
}
```

```
// VJS1365b.jsl
public class MyClass
{
    /** @property */
    void set_Val(int i)
    {
        value = i;
    }

    public static void main(String [] args)
    {
        MyClass c = new MyClass();
        c.set_Val(10); // OK access using method syntax
        c.Val = 10;    // VJS1365 accessing a property using property syntax
    }

    private int value;
}
```

## Comentarios

Visual J# no asocia ninguna importancia especial a métodos de descriptor de acceso de propiedad. Se tratan como métodos de la clase en sí misma.

# Error del compilador VJS1366

[Compiler Error VJS1366]

**No se puede tener acceso al campo '*campo*'**

Había un campo inaccesible.

En el siguiente código se genera el error VJS1366:

```
// VJS1366.js1
class MyClass
{
    private int i;
}

class MyClass2
{
    public static void main()
    {
        MyClass x = new MyClass();
        x.i = 0;    // VJS1366, i is private
    }
}
```

# Error del compilador VJS1367

[Compiler Error VJS1367]

**No se puede tener acceso al campo heredado '*campo*' a través de este objeto**

No se pudo tener acceso a un campo heredado; hay que agregar la clase derivada al paquete de la clase base.

En el siguiente código se genera el error VJS1367:

```
// VJS1367a.jsl
// compile with: /target:library
package p;

public class Base
{
    protected int f;
}
public class Derived extends Base
{
}
```

```
// VJS1367b.jsl
// compile with: VJS1367a.jsl
// add MyClass to package p to resolve
class MyClass
{
    public static void main (String [] args)
    {
        int k = new p.Derived().f;    // VJS1367
    }
}
```

# Error del compilador VJS1368

[Compiler Error VJS1368]

## No se puede tener acceso al campo no estático '*campo*' desde la clase interna estática

No se puede tener acceso a miembros de instancia de una clase externa desde el ámbito de una clase interna estática.

En el siguiente código se genera el error VJS1368:

```
// VJS1368.js1
// compile with: /target:library
class MyClass
{
    int f = 2;
    static int f2 = 10;
    static class Inner
    {
        int k = f;    // VJS1368, f is an instance variable
        int k2 = f2;  // OK: f2 is a static member
    }
    class Inner2
    {
        int k = f;    // OK: access instance member from instance inner class
    }
}
```



# Error del compilador VJS1369

[Compiler Error VJS1369]

**No se puede tener acceso al método no estático '*método*' desde una clase interna estática**

No se puede tener acceso a miembros no estáticos desde una clase interna estática.

En el siguiente código se genera el error VJS1369:

```
// VJS1369.js1
// compile with: /target:library
class MyClass
{
    int instanceMethod()
    {
        return 1;
    }
    static int staticMethod()
    {
        return 2;
    }
    static class Inner
    {
        int k = instanceMethod();    // VJS1369
        int k2 = staticMethod ();    // OK
    }

    class Inner2
    {
        // OK: accessing an instance member from an instance inner class
        int k = instanceMethod ();
    }
}
```

# Error del compilador VJS1371

[Compiler Error VJS1371]

**No se puede tener acceso al campo '*campo*' si no está declarado aquí totalmente**

Un campo debe estar declarado para poder utilizarlo en un inicializador estático.

En el siguiente código se genera el error VJS1371:

```
// VJS1371.js1
class MyClass
{
    static
    {
        i = 2;    // VJS1371, declare i before this static initializer
    }

    static int i;

    public static void main()
    {
    }
}
```

# Error del compilador VJS1372

[Compiler Error VJS1372]

**El campo de interfaz '*campo*' debe contener un inicializador**

No se inicializó ningún campo de interfaz.

En el siguiente código se genera el error VJS1372:

```
// VJS1372.js1
interface Point
{
    int i;    // VJS1372
    // try the following line instead
    // int i = 0;
}

public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1373

[Compiler Error VJS1373]

## Falta el exponente en el literal de punto flotante

En un valor de punto flotante falta un exponente.

En el siguiente código se genera el error VJS1373:

```
// VJS1373.js1
class MyClass
{
    float f = 3.40282347e+f;    // VJS1373
    // try the following line instead
    // float f = 3.40282347e+38f;

    public static void main()
    {
    }
}
```

# Error del compilador VJS1374

[Compiler Error VJS1374]

**'literal' no es un valor válido para un literal de tipo 'float'**

El compilador detectó un valor de punto flotante mal formado.

En el siguiente código se genera el error VJS1374:

```
// VJS1374.jsl  
class MyClass  
{  
    float f = 11111111111111111111111111111111e+2f;    // VJS1374  
    // try the following line instead  
    // float f = 3.140e+30f;  
  
    public static void main()  
    {  
    }  
}
```

# Error del compilador VJS1377

[Compiler Error VJS1377]

**La variable local '*var*' debe ser final para utilizarla en una clase interna**

No se puede tener acceso a las variables locales no marcadas como **final** en una clase envolvente desde una clase interna.

En el siguiente código se genera el error VJS1377:

```
// VJS1377.js1
// compile with: /target:library
class MyClass
{
    void Test()
    {
        int k = 30;
        final int k2 = 20;
        class Inner
        {
            int p = k;    // VJS1377
            int p2 = k2;   // OK
        }
    }
}
```

# Error del compilador VJS1379

[Compiler Error VJS1379]

## No se puede utilizar super en un contexto estático

La palabra clave **super** se utilizó de forma incorrecta.

En el siguiente código se genera el error VJS1379:

```
// VJS1379.js1
class MyClass
{
    int i = 0;
}

class MyClass2 extends MyClass
{
    static void Test()
    {
        super.i = 1;    // VJS1379, super can't be used in static method
    }
}

class MyClass3
{
    public static void main()
    {
        new MyClass2().Test();
    }
}
```

# Error del compilador VJS1380

[Compiler Error VJS1380]

## No se permite aquí la instrucción **break**

Se encontró una instrucción **break** en una ubicación no válida.

En el siguiente código se genera el error VJS1380:

```
// VJS1380.js1
public class MyClass
{
    public static void main()
    {
        break;    // VJS1380
    }
}
```



# Error del compilador VJS1381

[Compiler Error VJS1381]

## Literal de carácter no válido: '*literal*'

El compilador detectó un literal de carácter mal formado.

En el siguiente código se genera el error VJS1381:

```
// VJS1381.js1
class MyClass
{
    public static void main()
    {
        char c = 'a ';    // VJS1381, invalid char
        // try the following line instead
        char c = 'a';
    }
}
```

# Error del compilador VJS1382

[Compiler Error VJS1382]

## Nueva línea no válida en literal de carácter

No se permiten líneas nuevas en un literal de carácter.

En el siguiente código se genera el error VJS1382:

```
// VJS1382.js1
// compile with: /target:library
class MyClass
{
    char s = '
f';    // VJS1382
    // try the following line instead
    // char s = 'f';
}
```

# Error del compilador VJS1383

[Compiler Error VJS1383]

**Falta una comilla de cierre en el literal de carácter: *literal***

El compilador detectó un literal de carácter mal formado.

En el siguiente código se genera el error VJS1383:

```
// VJS1383.js1
class MyClass
{
    public static void main()
    {
        char c = 'a;    // VJS1383
        // try the following line instead
        // char c = 'a';
    }
}
```

# Error del compilador VJS1384

[Compiler Error VJS1384]

**Falta una comilla de cierre en el literal de cadena: *literal***

El compilador detectó un literal de cadena mal formado.

En el siguiente código se genera el error VJS1384:

```
// VJS1384.js1
class MyClass
{
    public static void main()
    {
        String s = "test;    // VJS1384
        // try the following line instead
        // String s = "test";
    }
}
```

# Error del compilador VJS1385

[Compiler Error VJS1385]

## Escape octal no válido '*valor*' (debe ser <= 0377)

Se utilizó un valor octal que no se encontraba comprendido dentro del intervalo de valores válidos.

En el siguiente código se genera el error VJS1385:

```
// VJS1385.js1
// compile with: /target:library
class MyClass
{
    // Error: octal char sequence should be less than o equal to 377
    char s = '\456';    // VJS1385
    // try the following line instead
    // char c = '\375';
}
```

# Error del compilador VJS1386

[Compiler Error VJS1386]

## Secuencia de escape no válida '*secuencia*'

El compilador detectó una secuencia de escape no válida. Por ejemplo, los dígitos octales válidos son sólo de 0 a 7.

En el siguiente código se genera el error VJS1386:

```
// VJS1386.js1
// compile with: /target:library
class MyClass
{
    char s = '\8';    // VJS1386
    char c = '\7';    // OK
}
```

# Error del compilador VJS1387

[Compiler Error VJS1387]

## Secuencia de escape Unicode no válida '*secuencia*'

El compilador detectó una secuencia de escape Unicode no válida. El formato válido es **\u####**.

En el siguiente código se genera el error VJS1387:

```
// VJS1387.js1
// compile with: /target:library
class MyClass
{
    char c = '\ug000';    // VJS1387
    // try the following line instead
    char cc = '\u1000';
}
```

# Advertencia del compilador (nivel 1) VJS1393

[Compiler Warning (level 1) VJS1393]

**Los atributos de ensamblado sólo se pueden definir al principio del archivo después de las instrucciones package e import**

Se encontró un atributo de nivel de ensamblado en el archivo de código fuente después de las definiciones de tipo.

En el siguiente código se genera el error VJS1393:

```
// VJS1393.js1
// compile with: /w:1
public class MyClass
{
    public static void main()
    {
    }
}
/** @assembly System.Reflection.AssemblyTitle("") */ // VJS1393, move to top of file
```



# Advertencia del compilador (nivel 1) VJS1395

[Compiler Warning (level 1) VJS1395]

**Esta característica no está disponible cuando las extensiones del lenguaje .NET están deshabilitadas**

El uso de determinadas extensiones del lenguaje no es compatible con la opción `/x:net` del compilador, que deshabilita las extensiones del lenguaje compatibles con Common Language Runtime. Por ejemplo, los atributos se omiten.

En el siguiente código se genera el error VJS1395:

```
// VJS1395.js1
// compile with: /x:net /W:1
/** @assembly System.Reflection.AssemblyTitle("") */ // VJS1395
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1396

[Compiler Error VJS1396]

**No se puede asociar el atributo '*atributo*' porque es una clase abstracta**

No se puede utilizar un atributo basado en una clase abstracta.

En el siguiente código se genera el error VJS1396:

```
// VJS1396a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public abstract class MyAttribute : System.Attribute
{
    public MyAttribute ()
    {
    }
}
```

```
// VJS1396b.jsl
// compile with: /reference:VJS1396a.dll
/** @attribute MyAttribute() */ // VJS1396, MyAttribute is abstract
class MyClass
{
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1397

[Compiler Error VJS1397]

**El atributo '*atributo*' no se hereda de '*System.Attribute*'**

Se aplicó un atributo que se definió correctamente.

En el siguiente código se genera el error VJS1397:

```
// VJS1397a.cs
// compile with: /target:library
// a C# program
// [System.AttributeUsage(System.AttributeTargets.All)]
public abstract class MyAttribute // : System.Attribute
{
    public MyAttribute ()
    {
    }
}
```

```
// VJS1397b.jsl
// compile with: /reference:VJS1397a.dll
/** @attribute MyAttribute() */ // VJS1397, MyAttribute not attr. class
// uncomment the attribute support syntax in VJS1397a.cs to resolve
class MyClass
{
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1398

[Compiler Error VJS1398]

## El atributo '*atributo*' tiene un parámetro de constructor no constante

Todos los parámetros de un constructor de atributo deben ser constantes; no se pueden pasar variables como parámetros a estos constructores.

En el siguiente código se genera el error VJS1398:

```
// VJS1398a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute: System.Attribute {
    public MyAttribute (MyType t) {
    }
}

public class MyType {
}
```

```
// VJS1398b.jsl
// compile with: /target:library /reference:VJS1398a.dll
/** @attribute MyAttribute (new MyType ()) */ // VJS1398
// to resolve, do not use the attribute or change the attribute
// ctor to be compliant with the common language specification (CLS)
public class MyClass
{
}
```

# Error del compilador VJS1399

[Compiler Error VJS1399]

**El atributo '*atributo*' tiene un parámetro de constructor de matriz que no es unidimensional**

Las matrices con dos o más dimensiones no se pueden pasar como parámetros de atributo y seguir siendo compatibles con CLS (Common Language Specification).

En el siguiente código se genera el error VJS1399:

```
// VJS1399a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    public MyAttribute (int [][] a)
    {
    }
}
```

```
// VJS1399b.jsl
// compile with: /target:library /reference:VJS1399a.dll
/** @attribute MyAttribute (new int[][]{{1,2},{1,2,3}}) */ // VJS1399
// to resolve, do not use the attribute or change the attribute
// ctor to only take a single dimensional array or other CLS
// compliant type
class MyClass
{
}
```

# Error del compilador VJS1401

[Compiler Error VJS1401]

## Tipo incorrecto para parámetro de atributo

En Visual J#, los parámetros de atributo deben ser tipos compatibles con CLS (Common Language Specification).

En la siguiente tabla se muestran los tipos de CLS y sus correspondientes tipos de Java.

Tipo de CLS	Tipo de Java
System.Byte(byte)	ubyte
System.Char(char)	char
System.Boolean(bool)	Boolean
System.Int16(short)	short
System.Int32(int)	int
System.Int64(long)	long
System.Single(float)	float
System.Double(double)	double

En el siguiente código se genera el error VJS1401:

```
// VJS1401a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute: System.Attribute {
    public MyAttribute (sbyte t) {
    }
}
```

```
// VJS1401b.jsl
// compile with: /target:library /reference:VJS1401a.dll
/** @attribute MyAttribute ((byte)10) */ // VJS1401
// do not use attribute or change attribute ctor to take
// a CLS compliant type
class MyClass
{
}
```

# Error del compilador VJS1402

[Compiler Error VJS1402]

**El campo con nombre o el parámetro de la propiedad '*parámetro*' no se encontró en el tipo '*tipo*'**

No existe ningún parámetro con nombre.

En el siguiente código se genera el error VJS1402:

```
// VJS1402a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    public MyAttribute ()
    {
    }
    public int Version = 20;
}
```

```
// VJS1402b.jsl
// compile with: /target:library /reference:VJS1402a.dll
/** @attribute MyAttribute (NonExistingField = 10) */ // VJS1402
// try the following line instead
// /** @attribute MyAttribute (Version = 10) */
class MyClass
{
}
```

# Error del compilador VJS1403

[Compiler Error VJS1403]

**El campo con nombre o el parámetro de la propiedad '*parámetro*' no se puede asignar**

Un parámetro con nombre es de sólo lectura; no se puede inicializar.

En el siguiente código se genera el error VJS1403:

```
// VJS1403a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    public MyAttribute ()
    {
    }
    public readonly int Version = 20;
}
```

```
// VJS1403b.jsl
// compile with: /target:library /reference:VJS1403a.dll
/** @attribute MyAttribute (Version = 10) */ // VJS1403
// try the following line instead
// or remove readonly from field definition
// /** @attribute MyAttribute () */
class MyClass
{
}
```



# Error del compilador VJS1404

[Compiler Error VJS1404]

**No se puede asignar un valor de tipo '*tipo1*' a un campo de atributo de tipo '*tipo2*'**

A un campo se le asignó un valor del tipo incorrecto.

En el siguiente código se genera el error VJS1404:

```
// VJS1404a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    public MyAttribute ()
    {
    }
    public int Version = 20;
}
```

```
// VJS1404b.jsl
// compile with: /target:library /reference:VJS1404a.dll
/** @attribute MyAttribute (Version = "string") */ // VJS1404
// try the following line instead
// /** @attribute MyAttribute (Version = 10) */
class MyClass
{
}
```

# Error del compilador VJS1405

[Compiler Error VJS1405]

**El campo con nombre o el parámetro de la propiedad '*parámetro*' es estático**

Los parámetros de campo con nombre no pueden ser estáticos.

En el siguiente código se genera el error VJS1405:

```
// VJS1405a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    public MyAttribute()
    {
    }
    public static int Version = 20;
}
```

```
// VJS1405b.jsl
// compile with: /target:library /reference:VJS1405a.dll
/** @attribute MyAttribute (Version = 10) */ // VJS1405
// remove static from field definition
class MyClass
{
}
```

# Error del compilador VJS1406

[Compiler Error VJS1406]

**El campo con nombre o el parámetro de la propiedad '*parámetro*' no es público**

No se puede tener acceso a un parámetro con nombre privado.

En el siguiente código se genera el error VJS1406:

```
// VJS1406a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All)]
public class MyAttribute : System.Attribute
{
    private int Version = 1;

    public MyAttribute()
    {
        int i = Version;
    }
}
```

```
// VJS1406b.jsl
// compile with: /target:library /reference:VJS1406a.dll
/** @attribute MyAttribute (Version = 10) */ // VJS1406
// make Version a public field
class MyClass
{
}
```

# Error del compilador VJS1407

[Compiler Error VJS1407]

**System.AttributeUsageAttribute** se puede asociar únicamente a una clase que proceda de **System.Attribute**

**System.AttributeUsageAttribute** es un atributo especial asociado a cada atributo System o a cada atributo definido por el usuario. El tipo al que se aplica este atributo debe proceder de **System.Attribute**.

En Visual J#, no se puede adjuntar este atributo a ningún tipo, puesto que no se admiten atributos creados por el usuario.

En el siguiente código se genera el error VJS1407:

```
// VJS1407.jsl
// compile with: /target:library
/** @attribute System.AttributeUsage(System.AttributeTargets.All) */ // VJS1407
class MyClass
{
}
```

# Error del compilador VJS1409

[Compiler Error VJS1409]

**El atributo '*atributo*' no puede aparecer varias veces en la misma declaración**

Se adjuntó un atributo varias veces.

En el siguiente código se genera el error VJS1409:

```
// VJS1409a.cs
// compile with: /target:library
// a C# program
[System.AttributeUsage(System.AttributeTargets.All /*, AllowMultiple = true */) ]
public class MyAttr : System.Attribute
{
    public MyAttr()
    {
    }
}
```

```
// VJS1409b.jsl
// compile with: /target:library /reference:VJS1409a.dll
/** @attribute MyAttr() */
/** @attribute MyAttr() */ // VJS1409
// uncomment attribute in VJS1409a.cs
// or remove second occurrence of attribute here
class MyClass
{
}
```

# Error del compilador VJS1410

[Compiler Error VJS1410]

## El atributo '*atributo*' no es válido en esta declaración

Se aplicó un atributo a un constructor para el que no está diseñado.

Por ejemplo, **System.ObsoleteAttribute** se puede adjuntar únicamente a tipos y miembros de tipos. Sin embargo, en el ejemplo siguiente, se aplica a un ensamblado.

En el siguiente código se genera el error VJS1410:

```
// VJS1410.jsl
/** @assembly System.Obsolete ("obsolete") */    // VJS1410
class MyClass
{
}
```

# Error del compilador VJS1418

[Compiler Error VJS1418]

**La directiva `@conditional` es válida sólo en métodos que devuelvan void.**

La directiva **`@conditional`** no se puede utilizar en métodos que no devuelvan void.

En el siguiente código se genera el error VJS1418:

```
// VJS1418.js1
// compile with: /target:library
class MyClass
{
    /** @conditional(DEBUG) */    // VJS1418
    public int Test1()
    {
        return 0;
    }
    // try the following instead
    /** @conditional(DEBUG)*/
    public void Test2()
    {
    }
}
```

# Error del compilador VJS1420

[Compiler Error VJS1420]

## La creación de un delegado debe tener 1 ó 2 argumentos

Un constructor de delegado sólo toma 1 ó 2 parámetros. Los constructores válidos para delegados son:

- *MyDelegate(method)*
- *MyDelegate(Object,String)*

En el siguiente código se genera el error VJS1420:

```
// VJS1420.jsl
// compile with: /target:library
class MyClass
{
    delegate void MyDel();
    public void Test()
    {
        MyDel d = new MyDel(new MyClass (), "DelMethod", null);    // VJS1420

        // OK. param is a method
        MyDel d1 = new MyDel(DelMethod);

        // OK. passing object and the method name
        MyDel d2 = new MyDel(new MyClass (), "DelMethod");
    }
    public void DelMethod()
    {
    }
}
```



# Error del compilador VJS1421

[Compiler Error VJS1421]

## La creación de un delegado no puede incluir un cuerpo

El compilador de Visual J# no admite la creación de delegados.

En el siguiente código se genera el error VJS1421:

```
// VJS1421.js1
class MyDelegate extends System.Delegate // VJS1421
{
}
```

# Error del compilador VJS1422

[Compiler Error VJS1422]

**El segundo argumento de la creación del delegado '*delegado*' debe ser una cadena**

El segundo argumento de la creación de un delegado debe ser una cadena

En el siguiente código se genera el error VJS1422:

```
// delegate creation second arg must be a string
// VJS1422.jsl
// compile with: /target:library
class MyClass
{
    delegate void MyDel();
    public void Test()
    {
        MyDel d = new MyDel(new MyClass (), new Object());    // VJS1422

        // OK, passing object and the method string
        MyDel d2 = new MyDel (new MyClass (), "DelMethod");
    }

    public void DelMethod()
    {
    }
}
```

# Error del compilador VJS1423

[Compiler Error VJS1423]

**El argumento de la creación del delegado '*delegado*' debe asignar un nombre a un método**

Se esperaba un método al crear un delegado.

En el siguiente código se genera el error VJS1423:

```
// VJS1423.js1
// compile with: /target:library
class MyClass
{
    delegate void MyDelegate();
    public void Test()
    {
        MyDelegate d = new MyDelegate (new MyClass ());    // VJS1423
        MyDelegate d1 = new MyDelegate (new MyClass ().DelMethod);    // OK
    }
    public void DelMethod()
    {
    }
}
```

# Error del compilador VJS1424

[Compiler Error VJS1424]

## El tipo de valor devuelto de '*método*' debe ser '*tipo*'

El tipo de valor devuelto de un delegado no coincidía con el tipo de valor devuelto de un método que se estaba asociando con el delegado.

En el siguiente código se genera el error VJS1424:

```
// VJS1424.js1
// compile with: /target:library
class MyClass
{
    delegate void MyDelegate();
    public void Test1()
    {
        MyDelegate d = new MyDelegate (new MyClass ().Test2);    // VJS1424

        // try the following line instead
        MyDelegate d1 = new MyDelegate (new MyClass ().Test3);
    }
    public int Test2()
    {
        return 0;
    }
    public void Test3()
    {
    }
}
```

# Error del compilador VJS1425

[Compiler Error VJS1425]

**La firma del método '*método*' no coincide con la firma del delegado '*delegado*'**

La firma de un delegado no coincidía con la firma del método que se estaba asociando con el delegado.

En el siguiente código se genera el error VJS1425:

```
// VJS1425.js1
// compile with: /target:library
class MyClass
{
    delegate void MyDelegate(int i);
    public void Test1()
    {
        MyDelegate d = new MyDelegate(new MyClass ().Test2);    // VJS1425
        // try the following line instead
        MyDelegate d1 = new MyDelegate(new MyClass ().Test3);
    }

    void Test2(long l)
    {
    }
    void Test3(int i)
    {
    }
}
```

# Error del compilador VJS1426

[Compiler Error VJS1426]

## El método estático '*método*' no se puede utilizar para inicializar un delegado

No se pueden pasar métodos estáticos para crear un delegado. Sólo se permiten métodos de instancia.

En el siguiente código se genera el error VJS1426:

```
// VJS1426.js1
// compile with: /target:library
class MyClass
{
    delegate void MyDelegate();
    public void Test1()
    {
        MyDelegate d = new MyDelegate(Test2);    // VJS1426
        // try the following line instead
        MyDelegate d1 = new MyDelegate(Test3);
    }

    static void Test2()
    {
    }
    void Test3()
    {
    }
}
```

# Error del compilador VJS1427

[Compiler Error VJS1427]

**El argumento de la creación del delegado .NET '*delegado*' debe ser un literal de cadena**

Al crear delegados de .NET Framework, el segundo argumento debe ser un literal de cadena.

En el siguiente código se genera el error VJS1427:

```
// VJS1427a.cs
// compile with: /target:library
// a C# program
public delegate void MyDelegate();
```

```
// VJS1427b.jsl
// compile with: /target:library /reference:VJS1427a.dll
class MyClass
{
    public void Test1()
    {
        String s = "m1";
        MyDelegate d = new MyDelegate(new MyClass (), s);    // VJS1427
        // try the following line instead
        MyDelegate d1 = new MyDelegate(new MyClass (), "Test2");
    }
    void Test2()
    {
    }
}
```

# Error del compilador VJS1428

[Compiler Error VJS1428]

**El método '*método*' inicia '*excepción*', que no está en la cláusula throws del delegado**

Un método asignado a un delegado no puede iniciar una excepción si la declaración de delegado no especifica la excepción.

En el siguiente código se genera el error VJS1428:

```
// or declare the delegate to throw
// VJS1428.jsl
// compile with: /target:library
class MyClass
{
    delegate void MyDelegate();
    public void Test1()
    {
        MyDelegate d = new MyDelegate(new MyClass ().Test2);    // VJS1428
        // try the following line instead
        MyDelegate d1 = new MyDelegate(Test3);
    }

    void Test2() throws Exception
    {
    }
    void Test3()
    {
    }
}
```



# Advertencia del compilador (nivel 1) VJS1434

[Compiler Warning (level 1) VJS1434]

'**@attribute**' se omitirá (extensiones deshabilitadas)

Las palabras clave como **@dll.import**, **@attribute**, **@com** y **delegate** son extensiones del lenguaje Java estándar. Al compilar con la opción **/x:all**, se deshabilitan las extensiones. Se omite la extensión.

En el siguiente código se genera el error VJS1434:

```
// VJS1434.jsl
// compile with: /target:library /x:all /W:1
/** @dll.import("user32") */    // VJS1434
class MyClass
{
}
```

# Error del compilador VJS1435

[Compiler Error VJS1435]

**No se puede declarar un delegado dentro de la clase interna '*clase*'**

No se puede declarar un delegado como miembro de una clase interna.

En el siguiente código se genera el error VJS1435:

```
// VJS1435.js1
// compile with: /target:library

class MyClass
{
    class InnerClass
    {
        delegate void del();    // VJS1435
    }

    delegate void MyDelegate1();    // OK, declare as an outer class member
}

delegate void MyDelegate2 ();    // OK, declare delegate as separate member
```

# Error del compilador VJS1436

[Compiler Error VJS1436]

## No se permite una llamada a un método en un literal nulo

Una llamada a un método en la expresión nula dará siempre un error en tiempo de ejecución.

En el siguiente código se genera el error VJS1436:

```
// VJS1436.js1
class MyClass
{
    public static void main (String [] args)
    {
        null.toString();    // VJS1436, method call on null literal
    }
}
```

# Error del compilador VJS1437

[Compiler Error VJS1437]

## Aquí no se permiten llamadas a métodos

Las cadenas de constantes sólo se pueden pasar como parámetros.

En el siguiente código se genera el error VJS1437:

```
// VJS1437.js1
// compile with: /target:library
/** @assembly System.Reflection.AssemblyTitle("abc.ToString()") */ // VJS1437
// try the following line instead
// /** @assembly System.Reflection.AssemblyTitle("abc") */
class MyClass
{
}
```

# Error del compilador VJS1438

[Compiler Error VJS1438]

**No se puede tener acceso al miembro no estático '*nombre*' en un contexto estático**

No se puede tener acceso a los miembros de instancia en un constructor estático ni en una función miembro estática de esa clase.

En el siguiente código se genera el error VJS1438:

```
// VJS1438.js1
// compile with: /target:library
// VJS1438
class MyClass
{
    int name = 0;
    public static int function()
    {
        int i = 0;
        i = name;
        return i;
    }
}
```

# Error del compilador VJS1439

[Compiler Error VJS1439]

## El inicializador de la matriz no está correctamente estructurado

Las matrices mixtas no se inicializan correctamente. Se deben inicializar los elementos de la matriz por separado. El modo en que se hacen las inicializaciones de las matrices mixtas es incorrecto. Se deben inicializar los elementos de la matriz por separado.

En el siguiente código se genera el error VJS1439:

```
// VJS1439.jsl
// compile with: /target:library
class Myclass
{
    void f()
    {
        int [,][] arr = new int [10,10][10];    // VJS1439
    }
}
```

Utilice el siguiente código correcto en su lugar:

```
// VJS1439a.jsl
// compile with: /target:library
class Myclass
{
    void f()
    {
        // Remove the last dimension and explicitly initialize
        // the rectangular array.
        int [,][] arr = new int [10,10][];
        for (int i = 0; i < 100; i++)
            for (int j = 0; j < 100; j++)
                arr[i,j] = new int[10];
    }
}
```

# Error del compilador VJS1457

[Compiler Error VJS1457]

## La superclase o la interfaz '*tipo*' no es compatible con CLS

Un tipo del que deriva un tipo de Java no era compatible con Common Language Specification (CLS).

En el siguiente código se genera el error VJS1457:

```
// VJS1457a.cs
// compile with: /unsafe /target:library
public abstract class Unsafe
{
    // unsafe: not CLS compliant and there is no Java-language equivalent
    public abstract unsafe void UnsafeMethod(byte * src);
}
```

```
// vjs1457b.jsl
// compile with: /reference:vjs1457a.dll
public class MyClass extends Unsafe    // VJS1457
{
}
```

# Advertencia del compilador (nivel 1) VJS1466

[Compiler Warning (level 1) VJS1466]

**Se va a asociar el atributo `System.CLSCompliantAttribute`, pero el compilador no comprobará si el ensamblado es compatible con CLS**

Se utilizó **CLSCompliantAttribute** y el compilador de Visual J# lo omitió, por lo que no se valida la compatibilidad con Common Language Specification (CLS).

En el siguiente código se genera el error VJS1466:

```
// VJS1466.jsl
// compile with: /w:1
/** @attribute System.CLSCompliant (true) */ // VJS1466
public class MyClass
{
    public static void main (String [] args)
    {
    }
}
```



# Error del compilador VJS1467

[Compiler Error VJS1467]

**System.TypedReference** es un tipo válido sólo para variables locales y parámetros

Se utilizó **TypedReference** en un lugar no válido.

En el siguiente código se genera el error VJS1467:

```
// VJS1467.js1
class MyClass
{
    System.TypedReference typeRef;    // VJS1467
    public static void main (String [] args)
    {
        System.TypedReference localRef;    // OK on local var
    }
    public void m (System.TypedReference paramRef)    // OK on param
    {
    }
}
```

# Error del compilador VJS1468

[Compiler Error VJS1468]

**Se especificó un ensamblado o una versión de archivo '*versión*' no válidos en el atributo de ensamblado**

El formato del número de versión de un ensamblado es mayor.menor.compilación.revisión, y todos estos números deben ser mayores que 0.

En el siguiente código se genera el error VJS1468:

```
// VJS1468.js1
// compile with: /target:library
import System.Reflection.*;
/** @assembly AssemblyVersion ("1.-1.1.1") */    // VJS1468
class MyClass
{
}
```

# Error del compilador VJS1469

[Compiler Error VJS1469]

## El ensamblado no tiene firma retrasada

Había dos atributos en conflicto en su especificación por un ensamblado con firma retrasada; el atributo **AssemblyDelaySignAttribute** especificó que el ensamblado se debía firmar con retraso, pero el archivo de clave pasado al atributo **AssemblyKeyFileAttribute** tenía una clave pública y privada.

Utilice [sn /p](#) para crear un archivo de clave con una clave pública sólo.

En el siguiente código se genera el error VJS1469:

```
// VJS1469.jsl
import System.Reflection.*;
import System.Runtime.CompilerServices.*;
/** @assembly AssemblyDelaySign(false) */
/** @assembly AssemblyKeyFile("VJS1469.key") */ // VJS1469, only has public key
class MyClass
{
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1470

[Compiler Error VJS1470]

## Archivo de clave no válido: '*archivo*'

No se encontró en el disco el nombre de archivo pasado al atributo **AssemblyKeyFileAttribute**, no contenía un par de claves válido o no contenía una clave pública válida.

En el siguiente código se genera el error VJS1470:

```
// VJS1470.js1
import System.Reflection.*;
import System.Runtime.CompilerServices.*;

/** @assembly AssemblyKeyFile("file") */ // VJS1470
class MyClass
{
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1471

[Compiler Error VJS1471]

**Nombre de clave no válido: 'contenedor'**

El nombre de clave pasado al atributo **AssemblyKeyNameAttribute** no era válido.

En el siguiente código se genera el error VJS1471:

```
// VJS1471.js1
import System.Reflection.*;
import System.Runtime.CompilerServices.*;

/** @assembly AssemblyKeyName("container") */ // VJS1471
class MyClass
{
    public static void main (String [] args)
    {
    }
}
```

# Error del compilador VJS1472

[Compiler Error VJS1472]

**'*tipo1*' no es válido en declaraciones de campos o métodos, utilice '*tipo2*' en su lugar**

No se puede utilizar el tipo *tipo1* como declaración de campo o método. Para resolverlo, utilice *tipo2* en su lugar.

En el siguiente código se genera el error VJS1472:

```
// VJS1472.js1
// compile with /target:library
// VJS1472 expected
public class MyClass
{
    void f(System.Int32 int32) {}
}
```

# Advertencia del compilador (nivel 3) VJS1493

[Compiler Warning (level 3) VJS1493]

## La variable '*var*' no se utiliza

Se declaró una variable pero no se inicializó ni utilizó.

En el siguiente código se genera el error VJS1493:

```
// VJS1493.js1
// compile with: /W:3
public class MyClass
{
    public static void main()
    {
        int i;    // VJS1493, i is not initialized or used
    }
}
```

# Advertencia del compilador (nivel 1) VJS1499

[Compiler Warning (level 1) VJS1499]

**No se admite 'private protected', se utilizará 'protected'**

El especificador de acceso **private protected** no es compatible; hay que utilizar **protected** en su lugar.

En el siguiente código se genera el error VJS1499:

```
// VJS1499.js1
// compile with: /W:1
public class MyClass
{
    private protected void Test()    // VJS1499
    // try the following line instead
    // protected void Test()
    {
    }
    public static void main()
    {
    }
}
```



# Advertencia del compilador (nivel 1) VJS1500

[Compiler Warning (level 1) VJS1500]

**El tipo '*tipo*' se encuentra varios binarios importados. Se utilizará el que se encuentra en '*archivo*'**

Se definió un tipo en dos o más ensamblados a los que se hace referencia. El compilador de Visual J# eligió el tipo definido en el archivo ***archivo***.

En el siguiente código se genera el error VJS1500:

```
// VJS1500a.cs
// compile with: /target:library
// a C# program
public class MyClass
{
    public static int i = 1;
}
```

```
// VJS1500b.cpp
// compile with: /LD /clr
// a C++ program
#using <mscorlib.dll>
public __gc class MyClass
{
public:
    static int i = 0;
};
```

```
// VJS1500c.jsl
// compile with: /reference:VJS1500a.dll /reference:VJS1500b.dll /W:1
// VJS1500 expected
public class MyClass2
{
    public static void main()
    {
        System.Console.WriteLine(MyClass.i);
    }
}
```

# Advertencia del compilador (nivel 1) VJS1501

[Compiler Warning (level 1) VJS1501]

**El método '*método*' no es compatible con el método de la clase base '*método base*'. Dicha incompatibilidad se omitirá**

Esta advertencia aparece cuando el compilador detecta el reemplazo de un método en **System.Object** pero con una firma diferente. Esto puede ocurrir con el código heredado de Java.

En el siguiente código se genera el error VJS1501:

```
// VJS1501.jsl
// compile with: /W:1 /target:library
class MyClass
{
    public int GetType()    // VJS1501, GetType is in System.Object but signature differs
    // try the following line instead
    // public int GetMyType()
    {
        return 0;
    }
}
```

# Error del compilador VJS1502

[Compiler Error VJS1502]

**El ámbito de 'Package' se utilizará en los ensamblados con la opción de ámbito seguro. Se puede iniciar una excepción para el acceso no válido en tiempo de ejecución.**

Se utilizó la opción [/securescoping](#) del compilador en un ensamblado, al que hizo referencia y tuvo acceso otro ensamblado utilizando el mismo paquete. Si el llamador no estuviera en el mismo paquete, la llamada no sería válida.

En el siguiente código se genera el error VJS1502:

```
// VJS1502a.jsl
// compile with: /securescoping /target:library
package p;
public class MyClass
{
    void Test()
    {
    }    // package-scoped member
}
```

```
// VJS1502b.jsl
// compile with: /reference:VJS1502a.dll /target:library
package p;
public class b
{
    public void Test2(MyClass A)
    {
        A.Test();    // VJS1502 using package access
    }
}
```

# Advertencia del compilador (nivel 1) VJS1503

[Compiler Warning (level 1) VJS1503]

**'miembro' no está permitido**

Se invocó a un miembro marcado como no permitido.

En el siguiente código se genera el error VJS1503:

```
// VJS1503a.cs
// compile with: /target:library
// a C# program
public class VJS1503a
{
    [System.Obsolete ("deprecated")]
    public void Test()
    {
    }
}
```

```
// VJS1503b.jsl
// compile with: /target:library /reference:VJS1503a.dll /W:1
class MyClass
{
    public void Test()
    {
        new VJS1503a () .Test ();    // VJS1503
    }
}
```

# Advertencia del compilador (nivel 1) VJS1504

[Compiler Warning (level 1) VJS1504]

## Ya se importó implícitamente el paquete '*paquete*'

El compilador emite esta advertencia cuando se intenta importar un paquete que ya ha importado el compilador implícitamente. Para resolver esta advertencia, elimine la instrucción de importación explícita.

En el siguiente código se genera el error VJS1504:

```
// VJS1504.js1
// compile with: /target:library /W:1
import java.lang.*; // VJS1504 Package java.lang is implicitly imported
class MyClass
{
}
```

# Advertencia del compilador (nivel 1) VJS1505

[Compiler Warning (level 1) VJS1505]

**Ya se importó el paquete '*paquete*'**

Se importó un paquete (o espacio de nombres) más de una vez.

En el siguiente código se genera el error VJS1505:

```
// VJS1505.js1
// compile with: /W:1
import System.*;    // VJS1505
import System.*;
class Test
{
    public static void main(String[] args)
    {
    }
}
```

# Advertencia del compilador (nivel 4) VJS1506

[Compiler Warning (level 4) VJS1506]

**El sufijo 'l' se confunde fácilmente con el dígito '1'; hay que utilizar 'L' para mayor claridad**

Para evitar posibles errores de sintaxis, utilice la letra 'L' mayúscula para especificar un valor long en lugar de la letra 'l' minúscula.

En el siguiente código se genera el error VJS1506:

```
// VJS1506.jsl
// compile with: /W:4
class MyClass
{
    public static void TestL(long i)
    {
    }

    public static void TestL(int i)
    {
    }

    public static void main()
    {
        TestL(25l);    // VJS1506
        // try the following line instead
        // TestL(25L);
    }
}
```

# Advertencia del compilador (nivel 3) VJS1507

[Compiler Warning (level 3) VJS1507]

## El operador 'instanceof' será siempre true

El compilador detectó que una comprobación de **instanceof** será siempre correcta y devolverá true. Esto puede ocurrir cuando se utiliza el operador **instanceof** para comprobar si un objeto de clase derivada es una instancia de una clase base o una interfaz implementada. La advertencia aparece porque la comprobación es redundante.

En el siguiente código se genera el error VJS1507:

```
// VJS1507.js1
// compile with: /W:3
interface I { }

class A { }

class B extends A implements I { }

public class MyClass
{
    public static void main (String [] args)
    {
        B b = new B();

        // Each of the instanceof operators below give VJS1507

        if (b instanceof B)    // VJS1507 b is of type B
        {
        }

        if (b instanceof A)    // VJS1507 b is instance of B that extends A
        {
        }

        if (b instanceof I)    // VJS1507 b is instance of B that implements I
        {
        }

        if (b instanceof Object)    // VJS1507 all classes derive from Object
        {
        }
    }
}
```



# Advertencia del compilador (nivel 1) VJS1510

[Compiler Warning (level 1) VJS1510]

## Bloque switch vacío

Se detectó un bloque switch vacío.

En el siguiente código se genera el error VJS1510:

```
// VJS1510.js1
// compile with: /W:1
class MyClass
{
    public static void main()
    {
        int i = 6;
        switch(i)
        {
            // to resolve, add something to the switch block, for example:
            /*
            case (5):
                System.Console.WriteLine("5");
                return;

            default:
                System.Console.WriteLine("not 5");
                return;
            */
        } // VJS1510
    }
}
```

# Advertencia del compilador (nivel 3) VJS1512

[Compiler Warning (level 3) VJS1512]

**La variable '*var*' se ha inicializado, pero nunca se ha utilizado**

Se declaró e inicializó una variable, pero no se utilizó.

En el siguiente código se genera el error VJS1512:

```
// VJS1512.js1
// compile with: /W:3
class Test
{
    public static void main(String[] args)
    {
        int i = 0;    // VJS1512
    }
}
```

# Advertencia del compilador (nivel 1) VJS1513

[Compiler Warning (level 1) VJS1513]

**El tipo '*tipo*' se encuentra en un ensamblado importado y en el origen. Se utilizará éste último.**

El compilador detectó que un tipo del ensamblado importado estaba declarado también en el archivo de código fuente. Se utilizará el tipo del archivo de código fuente.

En el siguiente código se genera el error VJS1513:

```
// VJS1513.js1
// compile with: /target:library /W:1
package java.util;
class Vector { }    // VJS1513
```

# Error del compilador VJS1524

[Compiler Error VJS1524]

**La directiva @dll.import debe proporcionar un nombre de DLL**

El compilador detectó un especificación **@dll.import** mal formada.

En el siguiente código se genera el error VJS1524:

```
// VJS1524.js1
class MyClass
{
    /**
     * @dll.import()    // VJS1524, no .dll file specified
     // try the following line instead
     // * @dll.import("msvcrt", auto)
     */
    public static native int puts(System.String c);

    public static void main()
    {
    }
}
```

# Advertencia del compilador (nivel 3) VJS1525

[Compiler Warning (level 3) VJS1525]

**El nombre de parámetro '*parámetro1*' no coincide con el nombre de parámetro de la directiva @com.parameters correspondiente '*parámetro2*'**

Los nombres de parámetro deben coincidir entre la declaración del método y los parámetros especificados en un atributo **@com**.

En el siguiente código se genera el error VJS1525:

```
// VJS1525.js1
// compile with: /reference:TESTCOMOBJLib.dll /target:library /W:3
package testcomobj;

/** @com.class(classid=836A4F1C-AF14-4210-99DB-28C7BDD2DE2B,DynamicCasts) */
public class Calculator // contained in TEstCOMObjLib.dll
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return) */ // VJS1525
    public native int Add(int i, int b);
    // try the following line instead
    // public native int Add(int a, int b);
}
```

# Error del compilador VJS1526

[Compiler Error VJS1526]

**'return' debe estar en último lugar de la declaración @com.parameters**

El compilador detectó un identificador después de una instrucción return en una declaración de **@com.parameters**.

En el siguiente código se genera el error VJS1526:

```
// VJS1526.js1
// compile with: /target:library
class MyClass
{
    /** @com.parameters([type=I4] return, i) */    // VJS1526
    // try the following line instead
    // /** @com.parameters([type=I4] i, return) */
    native int method1(Object i);
}
```

# Error del compilador VJS1527

[Compiler Error VJS1527]

**'return' ya está definido en @com.parameters**

El compilador detectó una instrucción return duplicada en una declaración de **@com.parameters**.

En el siguiente código se genera el error VJS1527:

```
// VJS1527.js1
// compile with: /target:library
class MyClass
{
    /** @com.parameters([type=I4] i, return, return) */    // VJS1527
    // try the following line instead
    // /** @com.parameters([type=I4] i, return) */
    native int method1(int i);
}
```

# Error del compilador VJS1528

[Compiler Error VJS1528]

**La directiva @com.parameters tiene un número erróneo de parámetros para el miembro '*miembro*'**

El número de parámetros debe coincidir entre la declaración del método y los parámetros especificados en un atributo **@com**.

En el siguiente código se genera el error VJS1528:

```
// VJS1528.js1
// compile with: /target:library /reference:testcomobjlib.dll
// assumes the presence of a COM object, testcomobjlib.dll
/** @com.class(classid=836A4F1C-AF14-4210-99DB-28C7BDD2DE2B, DynamicCasts) */
public class Calculator
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=I4] a, [type=I4] return) */
    public native int Add(int a, int b);    // VJS1528
    // try the following line instead
    // public native int Add(int a);
}
```



# Error del compilador VJS1529

[Compiler Error VJS1529]

**Se requiere '*parámetro*' para '*atributo*'**

Un atributo no estaba definido totalmente.

En el siguiente código se genera el error VJS1529:

```
// VJS1529.js1
// compile with: /target:library
/** @com.class() */    // VJS1529
// try the following line instead
// /** @com.class(classid=911CAED0-2957-11d1-A55E-00A0C90F26EE) */
public class Calculator
{
}
```

# Error del compilador VJS1530

[Compiler Error VJS1530]

**'@attribute' no es válido para esta declaración**

Se aplicó un atributo a un constructor para el que no está diseñado.

En el siguiente código se genera el error VJS1530:

```
// VJS1530.js1
/** @com.register (clsid=2148925C-234F-11D2-8C4E-00C04F8F3341, typelib=2148925B-234F-11D2-8C4E-00C04F8F3341) */ // VJS1530
interface MyInterface
{
}
```

# Error del compilador VJS1531

[Compiler Error VJS1531]

**No se puede encontrar la clase COM .NET con GUID '*guid*'**

No se resolvió una referencia a una clase.

En el siguiente código se genera el error VJS1531:

```
// VJS1531.js1
// compile with: /target:library
/** @com.class(classid=911CAED0-2957-11d1-A55E-00A0C90F26EE) */ // VJS1531
// To resolve, reference a COM object on which JActiveX or uuidgen was run
// and that has a class with the specified ID.
class MyClass
{
    /** @com.parameters([type=I4], return) */
    int method1(int i);
}
```

# Error del compilador VJS1532

[Compiler Error VJS1532]

**No se puede encontrar la interfaz COM .NET con GUID '*guid*'**

No se resolvió una referencia a una interfaz.

En el siguiente código se genera el error VJS1532:

```
// VJS1532.js1
// compile with: /target:library
/** @com.interface(iid=911CAED0-2957-11d1-A55E-00A0C90F26EE) */ // VJS1532
// to resolve, reference a COM object on which JActiveX or uuidgen was run
// and that has an interface with the specified ID
interface MyInterface
{
}
```

# Error del compilador VJS1533

[Compiler Error VJS1533]

**No se puede encontrar ningún método .NET que se corresponda con el método '*método*' de la clase o interfaz COM '*tipo*'**

Para compilar el contenedor de una interfaz con atributos generado por la herramienta JActiveX, debe hacer referencia a la DLL generada por TlbImp para la biblioteca de tipos. El compilador de Visual J# intenta encontrar una correspondencia para cada método de la interfaz en la interfaz correspondiente de la DLL generada por TlbImp. En este caso, el compilador no encontró un método que se correspondiera con el nombre y la firma del método encontrado en la interfaz con atributos **@com.interface**.

En el siguiente ejemplo se produce este error porque no se encuentra el método `Add1` en la interfaz `ICalculator`.

En el siguiente código se genera el error VJS1533:

```
// VJS1533.jsl
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;

// Dual interface ICalculator
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
public interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add1", addFlagsVtable=4)
        @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return) */
    public int Add1(int f, int b);    // VJS1533
}
```

# Error del compilador VJS1534

[Compiler Error VJS1534]

**La directiva @com.parameters del parámetro '*parámetro*' no especifica ningún valor para '*tipo*'**

La directiva **@com.parameters** no tiene ningún atributo de tipo para un parámetro.

En el siguiente código se genera el error VJS1534:

```
// VJS1534.js1
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
public interface ICalculator extends com.ms.com.IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in] a, [in,type=I4] b, [type=I4] return) */    // VJS1534
        // try the following line instead
        // @com.parameters([in, type=I4] a, [in,type=I4] b, [type=I4] return) */
    public int Add(int a, int b);
}
```

# Error del compilador VJS1535

[Compiler Error VJS1535]

**La directiva `@com.parameters` del parámetro '*parámetro*' tiene un tipo no compatible**

La directiva **`@com.parameters`** tenía un valor de tipo de `STRUCT` o `PTR` y el parámetro real no era `STRUCT`. En el ejemplo siguiente, el parámetro `a` es del tipo `Object` y no es un `STRUCT` (una clase con atributos de **`@com.struct`**).

En el siguiente código se genera el error VJS1535:

```
// VJS1535.js1
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;
// Dual interface ICalculator
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
interface ICalculator extends com.ms.com.IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=STRUCT] a, [in,type=I4] b, [type=I4] return) */    // VJS1535
    public int Add(Object a, int b);
}
```

# Error del compilador VJS1537

[Compiler Error VJS1537]

**La directiva `@com.parameters` del parámetro '*parámetro*' tiene un tipo `SAFEARRAY VT` no compatible**

El parámetro de la directiva `@com.parameters` es un `SAFEARRAY` (**`type=SAFEARRAY`**), pero el valor del atributo **`vt`** no es compatible.

Visual J# sólo admite los siguientes valores:

- VT\_I2 = 2
- VT\_I4 = 3
- VT\_R4 = 4
- VT\_R8 = 5
- VT\_CY = 6
- VT\_DATE = 7
- VT\_BSTR = 8
- VT\_DISPATCH = 9
- VT\_BOOL = 11
- VT\_VARIANT = 12
- VT\_UNKNOWN = 13
- VT\_I1 = 16
- VT\_UI1 = 17

En el siguiente código se genera el error VJS1537:

```
// VJS1537.jsl
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;
// Dual interface ICalculator
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
public interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=SAFEARRAY,vt=10] a, [in,type=I4] b, [type=I4] return) */    //
    VJS1537
    public int Add(com.ms.com.SafeArray a, int b);
    // don't use Add or change its definition in the COM object
    // to use a different VT type
}
```



# Error del compilador VJS1538

[Compiler Error VJS1538]

**La directiva @com.parameters de '*parámetro*' utiliza un cálculo de referencias personalizado no compatible**

La directiva **@com.parameters** indica que se está utilizando un cálculo de referencias personalizado (**type=CUSTOM**). Esto no se admite en Visual J#.

En el siguiente código se genera el error VJS1538:

```
// VJS1538.js1
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;

// Dual interface ICalculator
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
public interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=CUSTOM,customMarshal="myMarshallerClass"] a, [in,type=I4] b,
    [type=I4] return) */    // VJS1538
    public int Add(int a, int b);
}
```

# Error del compilador VJS1539

[Compiler Error VJS1539]

## No se puede crear el archivo de código fuente intermedio para el contenedor de la interfaz COM

El compilador crea archivos temporales durante la compilación de archivos de código fuente de Java que tienen atributos **@com**. Este error se produce si el compilador no puede crear un archivo temporal. Los archivos temporales se crean en el directorio temporal del usuario o en el directorio actual.

Este error se provoca si no tiene permiso de escritura en el directorio temporal (o en el directorio actual) o si el disco está lleno.

# Error del compilador VJS1540

[Compiler Error VJS1540]

**En @dll o @com, '*parámetro*' no es válido para '*clase*'**

Se encontró un parámetro no válido en un atributo.

En el siguiente código se genera el error VJS1540:

```
// VJS1540.js1
// compile with: /target:library
/** @dll.import("test.dll", entrypoint="Test") */ // VJS1540
// try the following line instead
// /** @dll.import("test.dll") */
public class MyClass
{
}
```

# Error del compilador VJS1541

[Compiler Error VJS1541]

**En @dll o @com, '*miembro*' requiere '*parámetro*'**

Había un atributo mal formado porque faltaba información.

En el siguiente código se genera el error VJS1541:

```
// VJS1541.js1
// compile with: /target:library
// VJS1541 expected
/** @dll.struct(noAutoOffset) */
public class MyClass
{
    /** @dll.structmap() */
    // Try the following line instead:
    // /** @dll.structmap([offset=0]) */
    int i;
}
```

# Error del compilador VJS1542

[Compiler Error VJS1542]

**'pack' y 'noAutoOffset' no se pueden especificar conjuntamente para '*atributo*'**

Se especificaron parámetros excluyentes mutuamente para un atributo.

En el siguiente código se genera el error VJS1542:

```
// VJS1542.js1
// compile with: /target:library
/** @dll.struct(noAutoOffset, pack=1) */    // VJS1542
// try the following line instead
// /** @dll.struct(noAutoOffset) */
public class MyClass
{
}
```

# Error del compilador VJS1543

[Compiler Error VJS1543]

## Error de sintaxis en '@*atributo*' para la declaración de GUID

Se especificó un GUID incorrectamente.

En el siguiente código se genera el error VJS1543:

```
// VJS1543.js1
// compile with: /target:library
/** @com.class(classid=x) */ // VJS1543, add real classid
public class MyClass
{
}
```

# Error del compilador VJS1544

[Compiler Error VJS1544]

## Error de sintaxis en la declaración '@attribute'

Se encontró un error de sintaxis en una declaración de atributo.

En el siguiente código se genera el error VJS1544:

```
// VJS1544.js1
class MyClass
{
    /**
     * @dll.import(,auto)    // VJS1544, no .dll file specified
     // try the following line instead
     // * @dll.import("msvcrt",auto)
     */
    public static native int puts(System.String c);

    public static void main()
    {
    }
}
```

# Error del compilador VJS1545

[Compiler Error VJS1545]

**@dll.import** requiere que '*método*' sea nativo

Un método declarado con **@dll.import** se debe declarar con el calificador **native**.

En el siguiente código se genera el error VJS1545:

```
// VJS1545.js1
// compile with: /target:library
class MyClass
{
    /**
     * @dll.import("msvcrt", auto)
     */
    public static int puts(System.String c);    // VJS1545
    // try the following line instead
    // public static native int puts(System.String c);

    public static void main()
    {
    }
}
```



# Error del compilador VJS1546

[Compiler Error VJS1546]

**El tipo de .NET correspondiente para la clase o interfaz '*tipo*' tiene el mismo nombre de paquete. Utilice la opción /namespace o /out de tlbimp.exe para importar la biblioteca de tipos a un paquete diferente.**

Una interfaz con atributos **@com.interface** tiene el mismo nombre de paquete que la clase correspondiente generada por TlbImp.exe.

En el ejemplo siguiente, la clase correspondiente de .NET Framework de la DLL generada por TlbImp TESTCOMOBJLib.dll tiene también el mismo nombre de paquete. Para resolver este error, hay que utilizar `tlbimp /namespace=SomethingElse TestComObj.dll` y volver a compilar.

En el siguiente código se genera el error VJS1546:

```
// VJS1546.jsl
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
package TESTCOMOBJLib;
import com.ms.com.IUnknown;

// Dual interface ICalculator
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */ //
VJS1546
public interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return) */
    public int Add(int a, int b);
}
```

# Error del compilador VJS1547

[Compiler Error VJS1547]

**El campo correspondiente '*campo*' de .NET, requerido para J/Direct, no se encuentra en la clase '*tipo*'**

Para compilar el contenedor de una interfaz con atributos **@com** generado por la herramienta JActiveX, hay que hacer referencia a la DLL generada por TlbImp para la biblioteca de tipos. El compilador de Visual J# intenta encontrar una coincidencia para cada método de la interfaz en la interfaz correspondiente de la DLL generada por TlbImp. Aquí no se encontró el campo correspondiente en la clase generada por TlbImp.

En el ejemplo siguiente, hay que resolver el error quitando la declaración de campo o asignándola a una de las declaraciones de campo de la clase `TESTCOMOBJLib.MyStruct`.

En el siguiente código se genera el error VJS1547:

```
// VJS1547.js1
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
package testcomobj;

/** @com.struct(noAutoOffset) */

public final class MyStruct    // VJS1547
{
    /** @com.structmap([offset=0,type=I4] fld1) */
    public int fld1;

    /** @com.structmap([offset=4,type=I4] fld2) */
    public int fld2;
}
```

# Error del compilador VJS1548

[Compiler Error VJS1548]

**Utilice la opción `/jcpa:@file_name` para enlazar el tipo `tlbimported` correcto de .NET para '*tipo1*' que está en conflicto con el tipo '*tipo2*' de .NET.**

Hay dos instancias de la misma interfaz COM en dos bibliotecas diferentes. También hay dos interfaces de .NET Framework correspondientes en espacios de nombres diferentes (en sus propios ensamblados).

El compilador no puede resolver qué interfaz COM se corresponde con qué interfaz de .NET Framework. Hay que utilizar la opción `/jcpa` del compilador para especificar explícitamente esta correspondencia.

# Error del compilador VJS1549

[Compiler Error VJS1549]

**Falta la directiva @com.parameters para el método '*método*' de la interfaz COM '*interfaz*'**

Falta la directiva **@com.parameters** en un método COM de una interfaz COM con atributos **@com**.

En el siguiente código se genera el error VJS1549:

```
// VJS1549.js1
// compile with: /target:library /reference:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4) */
    // add the following line to the attribute block
    // @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return)
    public int Add(int a, int b);    // VJS1549
}
```

# Error del compilador VJS1550

[Compiler Error VJS1550]

La directiva **@com.parameters** no contiene información acerca de todos los parámetros del método '*método*' de la interfaz **COM** '*interfaz*'

La directiva **@com.parameters** no describe todos los parámetros.

En el siguiente código se genera el error VJS1550:

```
// VJS1550.js1
// compile with: /target:library /r:TESTCOMOBJLib.dll
// assumes the presence of a COM object, testcomobjlib.dll
import com.ms.com.IUnknown;
/** @com.interface(iid=7F707219-A772-4437-AD86-DBABD5F6BC33, thread=AUTO, type=DUAL) */
interface ICalculator extends IUnknown
{
    /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="Add", addFlagsVtable=4)
        @com.parameters([in,type=I4] a, [type=I4] return) */
        // try the following line instead
        // @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return) */
    public int Add(int a, int b);    // VJS1550
}
```

# Advertencia del compilador (nivel 1) VJS1551

[Compiler Warning (level 1) VJS1551]

**'*atributo*' no es válido aquí, se omitirá.**

Se aplicó un atributo en una declaración para la que no estaba diseñado. El compilador omitió el atributo.

En el siguiente código se genera el error VJS1551:

```
// VJS1551.js1
// compile with: /target:library /W:1
class MyClass
{
    //ERROR: dll struct can be used only on class declarations
    /** @dll.struct () */    // VJS1551
    int x;
}

/** @dll.struct () */    // OK
class MyStruct
{
}
```

# Error del compilador VJS1552

[Compiler Error VJS1552]

**'valor' no es un valor correcto para /jcpa**

Se pasó un valor no válido a la opción [/jcpa](#) del compilador.

En el siguiente código se genera el error VJS1552:

```
// VJS1552.js1
// compile with: /jcpa:#
// VJS1552 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Advertencia del compilador (nivel 2) VJS1553

[Compiler Warning (level 2) VJS1553]

**No se encontró la biblioteca de tipos COM para la clase '*clasecom*'. Las aplicaciones COM que solían obtener acceso a esta clase utilizando una biblioteca de tipos, no funcionarán.**

El atributo **@com.register** no especifica un GUID de una biblioteca de tipos o el GUID que se especificó no estaba registrado en el equipo.

En Visual J++ 6.0, puede tener acceso a un componente de Java o COM marcado con el atributo **@com.register** de clientes COM. Si los clientes COM obtuvieran acceso al componente de Java o COM utilizando la biblioteca de tipos generada para el componente por la herramienta VJREG, el cliente COM utilizaría la interfaz dispinterface expuesta en el componente y habría detectado los Displs de los miembros. No está garantizado que los Displs de este componente sean los mismos en .NET Framework. Esto puede dar lugar a que fallen los clientes COM existentes. Para evitar estos errores en tiempo de ejecución, el compilador debe generar los mismos Displs que utilizaron los clientes COM. Esta advertencia aparece cuando el compilador no puede cargar la biblioteca de tipos y, por tanto, no puede garantizar que los clientes COM que utilizan este objeto por medio de la interfaz dispinterface continúen funcionando correctamente en tiempo de ejecución.

Para solucionar este problema:

- Si no se agregó un atributo typelib, agréguelo a la clase **@com.register**. El valor del GUID de typelib debe ser el mismo que el del GUID asignado a la biblioteca de tipos generada por la herramienta VJREG.
- Hay que asegurarse de que la biblioteca de tipos con el GUID dado esté registrada en el equipo.

En el siguiente código se genera el error VJS1553:

```
// VJS1553.js1
// compile with: /target:library /W:2
// typelib attribute missing in @com.register directive
/** @com.register(clsid=b62107bb-18ed-4796-a61a-0ade6edd3a91) */
public class test{} // VJS1553
// typelib attribute found but type library is not registered on machine
/** @com.register(clsid=b62107bb-18ed-4796-a61a-0ade6edd3a91, typelib=84b1fa15-6a14-4663-be9e-aa23fe40090d) */
public class test2{} // VJS1553
```



# Error del compilador VJS1554

[Compiler Error VJS1554]

**DllImportAttribute** requiere que el método '*método*' sea estático y nativo

**DllImportAttribute** se puede asociar sólo a métodos que sean estáticos y nativos.

En el siguiente código se genera el error VJS1554:

```
// VJS1554.js1
import System.Runtime.InteropServices.*;

public class MyClass {
    /** @attribute DllImportAttribute("Test") */
    public static void Compute();    // VJS1554
    // try the following line instead
    // public static native void Compute();
}
```

# Error del compilador VJS1555

[Compiler Error VJS1555]

**La clase '*clase*' no se puede exponer a COM cuando no es una clase pública**

Sólo las clases públicas se pueden exponer a COM utilizando el atributo **@com.register**.

Este error se produce también cuando se intentan exponer clases privadas anidadas a COM por medio de una interfaz COM de la clase.

En el siguiente código se genera el error VJS1555:

```
// VJS1555a.jsl
// compile with: /target:library
/** @com.register(clsid=17d49bf0-d80f-4cb4-a40f-a17b1037bee5) */
class MyClass // VJS1555
// try the following line instead
// public class MyClass
{
}
```

```
// VJS1555b.jsl
import com.ms.com.*;
public class MyOuterClass
{
    private class MyEnum implements IEnumVariant // VJS1555
    {
        public IEnumVariant Clone() { return null; }
        public void Next(int celt, Variant[] rgvar, int[] pceltFetched) {}
        public void Reset() {}
        public void Skip(int celt) {}
    }
}
```

# Error del compilador VJS1566

[Compiler Error VJS1566]

**Dos entradas de datos de recursos del recurso de win32 tienen el mismo tipo, nombre y lenguaje. No se pueden asociar dichos recursos.**

El archivo de recursos especificado (.res) con [/win32res](#) está dañado, por lo que el formato es correcto pero los datos no.

Por ejemplo, si se adjunta un recurso a un archivo .res dos veces o se adjunta el archivo .res a sí mismo y, después, se intenta asociarlo utilizando la opción **/win32res**, daría lugar a un error:

```
copy some.res VJS1566.res
type some.res >> VJS1566.res
vjc /win32res:VJS1566.res test.java
```

En el siguiente código se genera el error VJS1566:

```
// VJS1566.jsl
// compile with: /win32res:VJS1566.res
// VJS1566 expected
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1561

[Compiler Error VJS1561]

**'*archivo*': el formato de archivo no cumple con el formato de archivo de win32**

Se pasó un archivo a la opción [/win32res](#) del compilador que no tenía el formato de archivo de recursos Win32.

En el siguiente código se genera el error VJS1561:

```
// VJS1561.js1
// compile with: /win32res:VJS1561.doc
// VJS1561 expected
// assume VJS1561.doc is a file on disk
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1562

[Compiler Error VJS1562]

**El archivo '*archivo*' no se pudo encontrar ni leer**

El compilador no pudo encontrar o leer un archivo pasado a [/win32res](#), [/resource](#) o [/linkresource](#).

En el siguiente código se genera el error VJS1562:

```
// VJS1562.js1
// compile with: /win32res:x.res
// VJS1562 expected
// assume x.res is not on disk
public class MyClass
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS1575

[Compiler Error VJS1575]

## No se puede llamar al constructor recursivamente (directa o indirectamente)

Este error se produce cuando se intenta realizar una llamada recursiva a un constructor dentro de un constructor.

En el siguiente código se genera el error VJS1575:

```
// VJS1575.js1
// compile with: /target:library
public class MyClass
{
    MyClass()
    {
        this();    // VJS1575, recursive call to the same constructor
    }
}
```

# Error del compilador VJS1576

[Compiler Error VJS1576]

## **No se puede crear el directorio para los archivos de código fuente temporales del compilador**

El compilador crea un directorio temporal durante la compilación. Este error se produce si el compilador no puede crear un directorio temporal.

Los motivos pueden ser que no tenga permiso de escritura en el directorio o que el disco esté lleno.

# Error del compilador VJS1578

[Compiler Error VJS1578]

## No se puede obtener el directorio de sistema de Visual J#

Este error se produce cuando la instalación de Visual J# está dañada. Puede que se eliminaran algunos directorios o que el Registro esté dañado.

Para resolver este error, repare o reinstale Visual J#.



# Error del compilador VJS1579

[Compiler Error VJS1579]

## Compilación cancelada por el usuario

Se seleccionó **Cancelar** en el menú **Generar** del entorno de desarrollo mientras había una compilación en curso.

# Error del compilador VJS1580

[Compiler Error VJS1580]

## Error interno del compilador: memoria insuficiente

El compilador se ha quedado sin memoria mientras compilaba los archivos de código fuente. Intente de nuevo la compilación después de realizar lo siguiente:

- Liberar memoria cerrando una o más aplicaciones que se estén ejecutando.
- Aumentar el tamaño del archivo pagefile de memoria virtual. Consultar la ayuda en pantalla del sistema operativo para saber cómo hacer esto en su sistema o ponerse en contacto con el administrador de su sistema.

# Error del compilador VJS1581

[Compiler Error VJS1581]

## Error interno del compilador: *razón*

Intente determinar si se están produciendo errores en el compilador debido a que no puede analizar una sintaxis inesperada. A continuación, póngase en contacto con el [Servicio de soporte técnico de Microsoft](#).

# Error del compilador VJS1582

[Compiler Error VJS1582]

## Error al asociar el atributo '*tipo*': '*mensaje*'

El compilador no pudo asociar el atributo al miembro determinado.

En el siguiente código se genera el error VJS1582:

```
// VJS1582.js1
// compile with: /target:library
// VJS1582 expected
/** @attribute System.Runtime.InteropServices.GuidAttribute("") */
class MyClass
{
}
```

# Error del compilador VJS1800

[Compiler Error VJS1800]

**El descriptor de acceso de la propiedad debe comenzar con get\_ o set\_**

Había un nombre de propiedad mal formado.

En el siguiente código se genera el error VJS1800:

```
// vjs1800.js1
// compile with: /target:library
public class C
{
    /** @property */
    int Val()    // VJS1800
    // try the following line instead
    // int get_Val()
    {
        return 0;
    }
}
```

# Error del compilador VJS1801

[Compiler Error VJS1801]

## El nombre de propiedad '*nombre*' no es un identificador válido

El nombre definido de una propiedad aparecerá como el nombre menos el prefijo `get_` o `set_` cuando lo utilice un cliente. Determinados caracteres no son válidos al principio de un identificador. El primer carácter después del prefijo `get_` o `set_` en el nombre de una propiedad debe ser un carácter válido con el que comenzar un identificador. Por ejemplo, un nombre de propiedad no puede comenzar con un número.

En el siguiente código se genera el error VJS1801:

```
// VJS1801.js1
// compile with: /target:library
public class MyClass
{
    /** @property */
    int get_2Val()    // VJS1801
    // try the following line instead
    // int get_Val()
    {
        return 0;
    }
}
```

# Error del compilador VJS1802

[Compiler Error VJS1802]

**'*identificador*' está definido de nuevo como nombre de propiedad o evento**

El nombre de propiedad definido en el archivo de código fuente de Visual J# aparece como el nombre menos `get_` o `set_` para los usuarios. Por tanto, no se puede definir un identificador para que sea el mismo que el nombre que mostrará una propiedad para los consumidores del componente.

En el siguiente código se genera el error VJS1802:

```
// VJS1802.js1
// compile with: /target:library
public class MyProp
{
    private int SomeNum = -1;
    // try the following line instead
    // private int i = -1;

    /** @property */
    public int get_SomeNum()    // VJS1802, SomeNum already defined
    {
        return i;
    }
}
```

# Error del compilador VJS1803

[Compiler Error VJS1803]

## El tipo de una propiedad no puede ser void

El descriptor de acceso get\_ de una propiedad debe devolver un valor.

En el siguiente código se genera el error VJS1803:

```
// VJS1803.js1
// compile with: /target:library
public class MyClass
{
    private int ival;

    /** @property */
    public void get_Val()    // VJS1803
    // try the following line instead
    // public int get_Val()
    {
        return 0;
    }
}
```



# Error del compilador VJS1804

[Compiler Error VJS1804]

## El método del descriptor de acceso set debe tener el tipo de valor devuelto void

El descriptor de acceso set de una propiedad no tenía el tipo de valor devuelto void.

En el siguiente código se genera el error VJS1804:

```
// VJS1804.js1
// compile with: /target:library
public class MyClass
{
    private int ival;

    /** @property */
    public int set_Val(int i)    // VJS1804
    // try the following line instead
    // public void set_Val(int i)
    {
        ival = i;
    }
}
```

# Error del compilador VJS1805

[Compiler Error VJS1805]

**Ambos métodos de descriptor de acceso de una propiedad o un evento deben tener los mismos modificadores**

Los métodos de descriptor de acceso de una propiedad o un evento deben utilizar los mismos modificadores de acceso.

En el siguiente código se genera el error VJS1805:

```
// vjs1805.js1
// compile with: /target:library
public class MyClass
{
    private int ival;

    /** @property */
    public int get_Val()
    {
        return ival;
    }

    /** @property */
    private void set_Val(int i)    // VJS1805
    // try the following line instead
    // public void set_Val(int i)
    {
        ival = i;
    }
}
```

# Error del compilador VJS1806

[Compiler Error VJS1806]

**El evento '*evento*' debe tener los descriptores de acceso add y remove**

Había un evento que no estaba definido totalmente.

En el siguiente código se genera el error VJS1806:

```
// vjs1806.js1
// compile with: /target:library
/** @delegate */
public delegate void ValueChanged();

public class Container    // VJS1806
{
    public ValueChanged ev = null;

    /** @event */
    public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    // event with add accessor also needs remove accessor
    // /** @event */
    // public void remove_Event(ValueChanged p)
    // {
    //     ev = (ValueChanged) System.Delegate.Remove(ev, p);
    // }
}
```

# Error del compilador VJS1807

[Compiler Error VJS1807]

## El nombre de evento '*identificador*' no es un identificador válido

El nombre definido de un evento aparecerá como el nombre menos el prefijo **add\_** o **remove\_** cuando lo utilice un cliente. Determinados caracteres no son válidos al principio de un identificador. El primer carácter después del prefijo **add\_** o **remove\_** en el nombre de un evento debe ser un carácter válido con el que comenzar un identificador. Por ejemplo, un nombre de evento no puede comenzar con un número.

En el siguiente código se genera el error VJS1807:

```
// vjs1807.jsl
// compile with: /target:library
/** @delegate */
public delegate void ValueChanged();

public class Container
{
    public ValueChanged ev = null;

    /** @event */
    public void add_2Event(ValueChanged p)    // VJS1807
    // try the following line instead
    // public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    /** @event */
    public void remove_2Event(ValueChanged p) // VJS1807
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```

# Error del compilador VJS1808

[Compiler Error VJS1808]

## El tipo de evento '*identificador*' no es un delegado

Se utilizó un identificador como delegado, pero no se declaró como tal.

En el siguiente código se genera el error VJS1808:

```
// vjs1808.js1
// compile with: /target:library
// uncomment the following line to resolve
// /** @delegate */
public delegate void ValueChanged();

public class Container
{
    public ValueChanged ev = null;

    /** @event */
    public void add_Event(ValueChanged p)    // VJS1808
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    /** @event */
    public void remove_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```

# Error del compilador VJS1809

[Compiler Error VJS1809]

## El descriptor de acceso de un evento debe tener el tipo de valor devuelto void

El descriptor de acceso de un evento tenía un tipo de valor devuelto no válido.

En el siguiente código se genera el error VJS1809:

```
// vjs1809.js1
// compile with: /target:library
/** @delegate */
public delegate void ValueChanged();

public class Container
{
    public ValueChanged ev = null;

    /** @event */
    public int add_Event(ValueChanged p)    // VJS1809
    // try the following line instead
    // and delete the return statement from the body of the event
    // public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
        return 0;
    }

    /** @event */
    public void remove_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```

# Error del compilador VJS1810

[Compiler Error VJS1810]

**El descriptor de acceso de un evento debe ser de tipo 'void add\_name(type)' o 'void remove\_name(type)', donde 'type' derive de System.Delegate**

El descriptor de acceso de un evento tiene un número de argumentos no válidos.

En el siguiente código se genera el error VJS1810:

```
// vjs1810.jsl
// compile with: /target:library
/** @delegate */
public delegate void ValueChanged();

public class Container
{
    public ValueChanged ev = null;

    /** @event */
    public void add_Event(ValueChanged p, int i)    // VJS1810
    // try the following line instead
    // public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    /** @event */
    public void remove_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```

# Error del compilador VJS1811

[Compiler Error VJS1811]

**El descriptor de acceso de un evento debe comenzar con add\_ o remove\_**

El nombre del descriptor de acceso de un evento no comenzaba como se esperaba.

En el siguiente código se genera el error VJS1811:

```
// vjs1811.js1
// compile with: /target:library
/** @delegate */
public delegate void ValueChanged();

public class Container
{
    public ValueChanged ev = null;

    /** @event */
    public void set_Event(ValueChanged p)    // VJS1811
    // try the following line instead
    // public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    /** @event */
    public void remove_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```



# Error del compilador VJS2003

[Compiler Error VJS2003]

## Se esperaba la directiva #endif

El editor de texto detectó que faltaba una directiva **#endif**.

En el siguiente código se genera el error VJS2003:

```
public class MyClass
{
    public static void main()
    {
        #if DEBUG
            // uncomment the following line to resolve
        // #endif
    }
}
```

# Error del compilador VJS2004

[Compiler Error VJS2004]

## Directiva de preprocesador inesperada

El editor de texto detectó una directiva inesperada.

En el siguiente código se genera el error VJS2004:

```
public class MyClass
{
    public static void main()
    {
        #endif    // no #if
    }
}
```

# Error del compilador VJS2008

[Compiler Error VJS2008]

**Límite del IDE excedido: el archivo no puede tener más de *número* líneas**

En la versión actual, este error se genera si el archivo de código fuente excede de 2.097.151 líneas.

# Error del compilador VJS2009

[Compiler Error VJS2009]

**Límite del IDE excedido: la línea no puede tener más de *número* caracteres**

En la versión actual, este error se genera si el archivo de código fuente excede de 2.046 caracteres.

# Error del compilador VJS2010

[Compiler Error VJS2010]

## Se esperaba la directiva #endregion

Se encontró una directiva **#region**, pero no la directiva **#endregion**.

En el siguiente código se genera el error VJS2010:

```
#region
public class MyClass
{
    public static void main()
    {
    }
}
// #endregion
```

# Error del compilador VJS2012

[Compiler Error VJS2012]

## Expresión de preprocesador no válida

Una expresión del preprocesador debe ser de tipo **boolean**.

En el siguiente código se genera el error VJS2012:

```
#if 1    // VJS2012
// try the following line instead
#if true
#endif
```

# Error del compilador VJS2013

[Compiler Error VJS2013]

**El nombre de archivo especificado para #line es demasiado largo**

Cuando se pasa un nombre de archivo a la directiva **#line**, el nombre no debe tener más de 260 caracteres.

# Error del compilador VJS2015

[Compiler Error VJS2015]

**Se esperaba un nombre de archivo, un comentario de una línea o un carácter de fin de línea**

El analizador detectó una instrucción mal formada.

En el siguiente código se genera el error VJS2015:

```
#line 4 hello
// try the following line instead
#line 4 "myfilename"
```



# Error del compilador VJS2101

[Compiler Error VJS2101]

## Se esperaba )

Se detectó que faltaba un paréntesis de cierre.

En el siguiente código se genera el error VJS2101:

```
public class MyClass
{
    public static void main(
        // try the following line instead
        // public static void main()
        {
        }
    }
}
```

# Error del compilador VJS2102

[Compiler Error VJS2102]

## Se espera un valor constante

Se debe pasar una constante a una instrucción **case**.

En el siguiente código se genera el error VJS2102:

```
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            case :    // VJS2102
                // try the following line instead
                // case 0:
                default:
            }
        }
    }
}
```

# Error del compilador VJS2103

[Compiler Error VJS2103]

## Se esperaba }

Se detectó que faltaba una llave de cierre.

En el siguiente código se genera el error VJS2103:

```
public class MyClass
{
    public static void main()
    {
    }
// }
```

# Error del compilador VJS2104

[Compiler Error VJS2104]

## Se esperaba {

Se detectó que faltaba una llave de apertura.

En el siguiente código se genera el error VJS2104:

```
public class MyClass
{
    public static void main()
    // {
    }
}
```

# Error del compilador VJS2105

[Compiler Error VJS2105]

**Se esperaba ;**

Se detectó que faltaba un punto y coma (;).

En el siguiente código se genera el error VJS2105:

```
public class MyClass
{
    int i
}
```

# Error del compilador VJS2106

[Compiler Error VJS2106]

**Se esperaba un identificador y '*símbolo*' es una palabra clave**

Se detectó una palabra clave reservada donde se esperaba un identificador.

En el siguiente código se genera el error VJS2106:

```
public class MyClass
{
    int int;
}
```

# Error del compilador VJS2107

[Compiler Error VJS2107]

## Error sintáctico, se esperaba '*símbolo*'

Se esperaba un símbolo, pero no se encontró.

En el siguiente código se genera el error VJS2107:

```
public class MyClass
{
    public static void main(String[ args)    // VJS2107
    // try the following line instead
    // public static void main(String[] args)
    {
    }
}
```

# Error del compilador VJS2120

[Compiler Error VJS2120]

## Se esperaba un identificador

Se detectó que faltaba un identificador.

En el siguiente código se genera el error VJS2120:

```
public class MyClass
{
    int ;
}
```



# Error del compilador VJS2112

[Compiler Error VJS2112]

## Se esperaba un tipo o un constructor

Se detectó una declaración mal formada.

En el siguiente código se genera el error VJS2112:

```
public class MyClass
{
    public ()    // VJS2112
    {
    }
}
```

# Error del compilador VJS2113

[Compiler Error VJS2113]

## Un constructor no puede tener un tipo de valor devuelto

Un constructor no puede tener un tipo de valor devuelto

En el siguiente código se genera el error VJS2113:

```
public class MyClass
{
    public MyClass() []    // VJS2113, remove array notation
    {
    }
}
```

# Error del compilador VJS2116

[Compiler Error VJS2116]

## Se esperaba un inicializador de matriz

Había una declaración de matriz mal formada.

En el siguiente código se genera el error VJS2116:

```
class Test
{
    public static void main(String[] args)
    {
        int[] x = new int[];    // VJS2116
        // try the following line instead
        int[] x2 = new int[10];
    }
}
```

# Error del compilador VJS2117

[Compiler Error VJS2117]

## No se esperaba un inicializador de matriz aquí

No se puede inicializar una matriz en el contexto actual.

En el siguiente código se genera el error VJS2117:

```
public class Class1
{
    public static void main(String[] args)
    {
        Class1 cl = new Class1();
        cl.method(new int[2]{});    // VJS2117
        // try the following line instead
        // cl.method(new int[2]);
    }
}
```

# Error del compilador VJS2118

[Compiler Error VJS2118]

## Se esperaba un tipo

Este error se produce cuando se espera un tipo pero no se encuentra.

En el siguiente código se genera el error VJS2118:

```
class Test
{
    public static void main(String[] args)
    {
        Test x = new Test();
        if (x instanceof )    // VJS2118
        // try the following line instead
        // if (x instanceof Test)
        {
        }
    }
}
```

# Error del compilador VJS2119

[Compiler Error VJS2119]

## Un delegado de multidifusión debe tener el tipo de valor devuelto void

Había una declaración de delegado de multidifusión mal formada debido al tipo de valor devuelto.

En el siguiente código se genera el error VJS2119:

```
class MyClass
{
    public multicast delegate int MyDelegate();    // VJS2119
    // try the following line instead
    public multicast delegate void CorrectDelegate();
}
```

# Error del compilador VJS2202

[Compiler Error VJS2202]

**La palabra clave case o default debe ir delante del código en el bloque switch**

El compilador detectó una instrucción inesperada en un bloque switch.

En el siguiente código se genera el error VJS2202:

```
public class MyClass
{
    public static void main()
    {
        int i = 0;
        switch (i)
        {
            i:    // VJS2202, delete
            case 0:
            default:
        }
    }
}
```

# Error del compilador VJS2203

[Compiler Error VJS2203]

**Sólo se pueden utilizar las expresiones assignment, call, increment, decrement y new como instrucción de expresión**

Había una instrucción mal formada.

En el siguiente código se genera el error VJS2203:

```
public class MyClass
{
    public static void main()
    {
        i;    // VJS2203
    }
}
```



# Error del compilador VJS2204

[Compiler Error VJS2204]

## Modificador '*modificador*' duplicado

No se permiten modificadores de acceso duplicados en una declaración.

En el siguiente código se genera el error VJS2204:

```
public class MyClass
{
    public public static void main()    // VJS2204, two public modifiers
    {
    }
}
```

# Error del compilador VJS2207

[Compiler Error VJS2207]

**Sólo se pueden utilizar expresiones de inicializador de matriz para asignarlos a tipos de matriz.**

Se intentó una inicialización de matriz en una variable que no era un tipo de matriz.

En el siguiente código se genera el error VJS2207:

```
public class Class1
{
    public static void main(String[] args)
    {
        int IAmNotAnArray = {1, 2};    // VJS2207
        // try the following line instead
        int[] x = new int[]{1,2};
    }
}
```

# Error del compilador VJS2208

[Compiler Error VJS2208]

**Una clase o un método de interfaz deben tener un tipo de valor devuelto**

La declaración de un método no incluía un tipo de valor devuelto.

En el siguiente código se genera el error VJS2208:

```
public class MyClass
{
    public static main()    // VJS2208
    // try the following line instead
    // public static void main()
    {
    }
}
```

# Error del compilador VJS2210

[Compiler Error VJS2210]

**Se esperaba una clase, una interfaz o un delegado**

Se detectó un símbolo no válido.

En el siguiente código se genera el error VJS2210:

```
int i;    // VJS2210
```

# Error del compilador VJS2211

[Compiler Error VJS2211]

## Las clases internas no deben contener interfaces

Una clase interna contenía una interfaz y esto no está permitido.

En el siguiente código se genera el error VJS2211:

```
public class MyClass
{
    class MyInnerClass
    {
        interface MyInterface    // VJS2211
        {
        }
    }
}
```

# Error del compilador VJS2213

[Compiler Error VJS2213]

**El símbolo '*símbolo*' no es válido en la declaración de un miembro de interfaz o clase**

Se detectó un símbolo no válido en una clase o una interfaz.

En el siguiente código se genera el error VJS2213:

```
public class MyClass
{
    default;    // VJS2213
    public static void main()
    {
    }
}
```

# Error del compilador VJS2301

[Compiler Error VJS2301]

## Línea nueva en constante

Se encontró un carácter de línea nueva en un literal de carácter o de cadena.

En el siguiente código se genera el error VJS2301:

```
class MyClass
{
    char s = '
        f';
    // try the following line instead
    // char s = 'f';
}
```

# Error del compilador VJS2302

[Compiler Error VJS2302]

## Una constante integral está fuera del intervalo para int

Una constante integral no puede representarse como **int**. Este error se produce cuando una constante **int** causa un desbordamiento.

En el siguiente código se genera el error VJS2302:

```
class MyClass
{
    int i = 1000000000000; // VJS2302
}
```



## Error del compilador VJS2303

[Compiler Error VJS2303]

### Una constante está fuera del intervalo para valores de punto flotante

Una constante no se puede representar como **float**. Este error se produce cuando una constante de punto flotante causa un desbordamiento o subdesbordamiento.

En el siguiente código se genera el error VJS2303:

```
class MyClass
{
    float f = 111111111111111111111111111111111111e+2f;    // VJS2303
    // try the following line instead
    // float f = 3.140e+30f;

    public static void main()
    {
    }
}
```

# Error del compilador VJS2304

[Compiler Error VJS2304]

## Número no válido

Se detectó un formato de número no válido.

En el siguiente código se genera el error VJS2304:

```
public class Class1
{
    public static void main(String[] args)
    {
        int a = 0x;    // VJS2304
        // try the following line instead
        // int a = 0xa;
    }
}
```

# Error del compilador VJS2305

[Compiler Error VJS2305]

## Literal de carácter vacío

Se asignó un carácter vacío a una variable de tipo **char** y esto no está permitido.

En el siguiente código se genera el error VJS2305:

```
class MyClass
{
    public static void main()
    {
        char i = '';    // VJS2305
        // try the following line instead
        char j = 'a';
    }
}
```

# Error del compilador VJS2306

[Compiler Error VJS2306]

## Demasiados caracteres en literal de carácter

Se asignó más de un carácter a una variable de tipo **char** y esto no está permitido.

En el siguiente código se genera el error VJS2306:

```
class MyClass
{
    public static void main()
    {
        char i = 'ab';    // VJS2306
        // try the following line instead
        char j = 'a';
    }
}
```

# Error del compilador VJS2307

[Compiler Error VJS2307]

## Secuencia de escape no reconocida

Había un carácter mal formado.

En el siguiente código se genera el error VJS2307:

```
class MyClass
{
    char s = '\8';    // VJS2307, not a valid octal char
    char c = '\7';    // OK
}
```

# Error del compilador VJS2308

[Compiler Error VJS2308]

## Hay una constante fuera del intervalo para valores de double

Una constante no se puede representar como **double**. Este error se produce cuando una constante double causa un desbordamiento o subdesbordamiento.

En el siguiente código se genera el error VJS2308:

```
public class Class1
{
    public static void main(String[] args)
    {
        double d = 4e3045d;    // VJS2308, try a smaller value
    }
}
```

# Error del compilador VJS2309

[Compiler Error VJS2309]

## Una constante integral está fuera del intervalo para long

Una constante integral no se puede representar como **long**. Este error se producirá siempre que **long** cause un desbordamiento.

En el siguiente código se genera el error VJS2309:

```
class MyClass
{
    long s = 1000000000000000000000000L; // VJS2309
}
```

# Error del compilador VJS2401

[Compiler Error VJS2401]

## Una interfaz no puede ser final

Había una definición de interfaz con el modificador de acceso **final** y esto no está permitido.

En el siguiente código se genera el error VJS2401:

```
final interface MyInterface    // VJS2401
{
}
```



# Error del compilador VJS2402

[Compiler Error VJS2402]

**No se permite el modificador '*modificador*' para una clase**

Se utilizó un modificador en una declaración de clase para la que no estaba diseñado.

En el siguiente código se genera el error VJS2402:

```
volatile class MyClass    // VJS2402
{
}
```

# Error del compilador VJS2403

[Compiler Error VJS2403]

**No se permite el modificador '*modificador*' para una interfaz**

Se utilizó un modificador incorrectamente en una declaración de interfaz.

En el siguiente código se genera el error VJS2403:

```
volatile interface inter1    // VJS2403
{
}
```

# Error del compilador VJS2404

[Compiler Error VJS2404]

**No se permite el modificador '*modificador*' para un delegado**

Se aplicó un modificador incorrectamente a una declaración de delegado.

En el siguiente código se genera el error VJS2404:

```
volatile delegate void Del();    // VJS2404
```

# Error del compilador VJS2405

[Compiler Error VJS2405]

## No se puede definir una clase estática en una clase interna

No se puede definir una clase interna estática cuando la clase envolvente no es estática.

En el siguiente código se genera el error VJS2405:

```
class Class1
{
    class InnerClass    // not static
    {
        static class InnnerMostClass    // VJS2405
        {
        }
    }
}
```

# Error del compilador VJS2406

[Compiler Error VJS2406]

**No se permite el modificador '*modificador*' para una clase local**

Se utilizó un modificador incorrectamente en una declaración de clase local o interna.

En el siguiente código se genera el error VJS2406:

```
class Class1
{
    public static void main(String[] args)
    {
        abstract static class LocalClass    // VJS2406, static not allowed here
        {
        }
    }
}
```

# Error del compilador VJS2407

[Compiler Error VJS2407]

**No se permite el modificador '*modificador*' para un constructor**

Se aplicó un modificador (***modificador***) incorrectamente a un constructor.

En el siguiente código se genera el error VJS2407:

```
class Class1
{
    static public Class1()    // VJS2407, static not allowed here
    {
    }
}
```

# Error del compilador VJS2408

[Compiler Error VJS2408]

**No se permite el modificador '*modificador*' para un método**

Se aplicó un modificador (***modificador***) incorrectamente a una declaración de método.

En el siguiente código se genera el error VJS2408:

```
class Class1
{
    public volatile static void main(String[] args)    // VJS2408, volatile not allowed here
    {
    }
}
```

# Error del compilador VJS2409

[Compiler Error VJS2409]

**No se permite el modificador '*modificador*' para un campo**

Se aplicó un modificador (***modificador***) incorrectamente a una declaración de campo.

En el siguiente código se genera el error VJS2409:

```
class Class1
{
    synchronized int FieldCantBeSync;    // VJS2409
}
```



# Error del compilador VJS2410

[Compiler Error VJS2410]

**No puede haber modificadores '*modificador1*' y 'abstract'**

Se aplicó incorrectamente un modificador (***modificador***) junto con el modificador abstracto.

En el siguiente código se genera el error VJS2410:

```
// vjs2410.js1
// compile with: /target:library
public abstract final class Class1    // VJS2410
{
}
```

# Error del compilador VJS2411

[Compiler Error VJS2411]

**No se permite el modificador '*modificador*' para un método de una interfaz**

Se aplicó un modificador (***modificador***) incorrectamente a una declaración de método en una interfaz.

En el siguiente código se genera el error VJS2411:

```
interface inter1
{
    final void method();    // VJS2411
}
```

# Error del compilador VJS2413

[Compiler Error VJS2413]

**No se permite el modificador '*modificador*' para un campo de una interfaz**

Se aplicó un modificador (***modificador***) incorrectamente a una declaración de campo en una interfaz.

En el siguiente código se genera el error VJS2413:

```
interface inter1
{
    volatile int a;    // VJS2413
}
```

# Error del compilador VJS2414

[Compiler Error VJS2414]

**No pueden estar los dos modificadores '*modificador1*' y '*modificador2*'**

Se aplicaron dos modificadores excluyentes mutuamente a la misma declaración.

En el siguiente código se genera el error VJS2414:

```
class Class1
{
    final volatile int a;    // VJS2414
}
```

# Error del compilador VJS2416

[Compiler Error VJS2416]

**Una clase externa no puede tener el modificador '*modificador*'**

Se aplicó incorrectamente un modificador (***modificador***) a la declaración de una clase externa.

En el siguiente código se genera el error VJS2416:

```
protected class MyClass    // VJS2416
{
    public static void main()
    {
    }
}
```

# Error del compilador VJS2417

[Compiler Error VJS2417]

**Una interfaz externa no puede tener el modificador '*modificador*'**

Se aplicó incorrectamente un modificador (***modificador***) a la declaración de una interfaz externa.

En el siguiente código se genera el error VJS2417:

```
private interface inter    // VJS2417
{
}
```

# Error del compilador VJS1581

[Compiler Error VJS1581]

## Error interno del compilador: *razón*

Intente determinar si se están produciendo errores en el compilador debido a que no puede analizar una sintaxis inesperada. A continuación, póngase en contacto con el [Servicio de soporte técnico de Microsoft](#).

# Compatibilidad con el lenguaje

El compilador de Visual J# proporciona compatibilidad con las extensiones de Microsoft® de Visual J++® 6.0, incluidos delegados, J/Direct®, la compatibilidad del atributo **@com** con la interoperabilidad de Java y COM, la compilación condicional y la mayoría de los demás atributos, como **@security** y **@hidden**. Visual J# también es compatible con el uso de bibliotecas de clases de .NET Framework y definidas por el usuario escritas en otros lenguajes compatibles con .NET Framework, como Visual C# y Visual Basic.

## En esta sección

### [Compatibilidad con Visual J++ 6.0](#)

Enumera las extensiones de Microsoft en Visual J++ 6.0 que son totalmente compatibles con Visual J#.

### [Compatibilidad con el uso de clases de .NET Framework](#)

Proporciona un breve ejemplo para mostrar el modo en que Visual J# .NET puede utilizar clases de .NET Framework que cumplan con Common Language Specification (CLS).

### [Sintaxis para utilizar .NET Framework](#)

Proporciona vínculos a temas que muestran la sintaxis para utilizar características de Common Language Runtime de .NET Framework.

### [Características de .NET Framework no compatibles](#)

Enumera las características de Common Language Runtime de .NET Framework que no son compatibles con Visual J#.

### [Generar y compilar código fuente dinámicamente en Java](#)

Explica la implementación de CodeDOM para Java que proporciona Visual J#.

### [Problemas en la generación de un proxy para servicios Web](#)

Trata consideraciones relacionadas con la adición a una aplicación de Visual J# de una referencia a un servicio Web XML.

### [Comentarios sobre documentación en Visual J#](#)

Explica las características del entorno de desarrollo integrado que proporcionan compatibilidad con comentarios sobre documentación.

### [Jerarquías de excepciones de Visual J#](#)

Describe la relación existente entre las excepciones de Java y las de .NET Framework.

## Secciones relacionadas

### [Referencia](#)

Temas de referencia sobre el compilador de Visual J#, el Conversor binario (Jblmp.exe), la compatibilidad de bibliotecas y lenguajes, la actualización de proyectos de Visual J++ 6.0, la sintaxis de Visual J# para la biblioteca de clases de .NET Framework y las características del entorno de desarrollo.

### [Compatibilidad con bibliotecas de clases](#)

Enumera los paquetes compatibles y no compatibles, así como información específica de implementación para la compatibilidad con bibliotecas de clases.



# Compatibilidad con Visual J++ 6.0

Las extensiones de Microsoft en Visual J++ 6.0 que se enumeran en este tema son totalmente compatibles con Visual J#.

Para buscar información de referencia sobre las extensiones, vea la documentación de Visual J++ 6.0 o Microsoft SDK para Java en MSDN Online Library

(<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vjcore98/html/vjovrdocumentationmap.asp>).

- J/Direct (para marcar llamadas a DLL nativas)
  - **@dll.import**
  - **@dll.struct**
  - **@dll.structmap**
- Java y COM, por ejemplo:
  - **@com.class**
  - **@com.interface**
  - **@com.method**
  - **@com.parameter**
  - **@com.register**
- Compilación condicional
  - **#if, #elif, #endif, #define**
  - **#warning, #error**
  - **@conditional** (utilizado para marcar métodos cuyas llamadas se pueden quitar condicionalmente del código generado)
- Compatibilidad con delegados y delegados de multidifusión
  - Clases **com.ms.lang.Delegate** y **com.ms.lang.MulticastDelegate**
  - Palabras clave **delegate** y **multicast**
- Otras directivas @
  - **@security**
  - **@hidden**
  - **@deprecated**

Las siguientes características del compilador de Visual J++ 6.0, que son específicas de los códigos de bytes de Java, no están disponibles en el compilador de Visual J#:

- Capacidad para importar información de clase desde códigos de bytes de Java (archivos .class).
- Capacidad para generar archivos .class desde archivos de código fuente de Java.
- Compatibilidad con CLASSPATH para resolver clases a las que se hace referencia en tiempo de compilación; se utiliza la opción [/reference \(Importar metadatos\)](#) del compilador en su lugar.

## Vea también

[Compatibilidad con el lenguaje](#) | [Buscar información de referencia sobre Java y JDK](#)

# Compatibilidad con el uso de clases de .NET Framework

El compilador de Visual J# admite el uso de cualquier API de bibliotecas de .NET Framework que cumplan con Common Language Specification (CLS). A continuación se muestra el programa de ejemplo "Hola a todos", que utiliza clases de .NET Framework.

## Ejemplo

```
// vjc_consume_netfx.jsl
import System.*; // contains the Console class
public class Hello
{
    public static void main(String[] args)
    {
        Console.WriteLine("Hello, World!");
    }
}
```

## Salida

```
Hello, World!
```

## Vea también

[Compatibilidad con el lenguaje](#)

# Sintaxis para utilizar .NET Framework

En esta sección se explican las características del compilador de Visual J# que se utilizan para .NET Framework. Estas características están disponibles de manera predeterminada. No obstante, se puede utilizar la opción [/x \(Deshabilitar extensiones del lenguaje\)](#) del compilador para deshabilitar estas características.

En esta sección se supone que se conoce .NET Framework, el sistema de tipos comunes, los tipos de valor, los tipos de referencia, así como la conversión boxing y unboxing. Para entender estos conceptos, vea [Programar con .NET Framework](#).

En esta sección se tratan los siguientes temas:

- [Semántica de la jerarquía de objetos](#)
- [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)
  - [ubyte](#)
  - [Matrices rectangulares](#)
  - [Nuevas directivas de preprocesamiento](#)
  - [Asociar atributos](#)
  - [Utilizar palabras clave como identificadores](#)
  - [Llamar a métodos que toman argumentos por referencia](#)
- [Definir y utilizar propiedades](#)
- [Definir y utilizar eventos](#)
- [Definir y utilizar delegados de .NET Framework](#)
- [Utilizar tipos de valor](#)
- [Utilizar tipos de valor correspondientes a tipos primitivos](#)
- [Utilizar enumeraciones](#)
- [Ejemplo: propiedades y eventos](#)
- [Llamar a métodos nativos](#)
- [Operadores de conversión de tipos](#)

## Vea también

[Compatibilidad con el lenguaje](#)

# Semántica de la jerarquía de objetos

En Visual J#, **java.lang.Object** se trata como una raíz específica del lenguaje de la jerarquía de objetos. El medio para tener acceso a este objeto desde lenguajes compatibles con Common Language Runtime es a través de **System.Object**. Por tanto, estos lenguajes no ven los objetos de Visual J# como objetos especiales que derivan de **java.lang.Object**.

Por ejemplo:

```
class A extends java.lang.Object {  
    public String toString() { return "Instance of A";}  
}
```

Se trata de una clase escrita en Visual J# y no se puede tener acceso al método **toString** del objeto en otros lenguajes. En su lugar, utilice el siguiente código para obtener la representación de la cadena del objeto desde Visual C#:

```
string str = new A().ToString();
```

**Nota** Esto es verdadero incluso si **A** reemplazara una implementación del método `toString`.

Para un programador de Java, Visual J# trata **System.Object** como **java.lang.Object** y establece una correspondencia entre sus funcionalidades de forma que cualquier diferencia sea transparente. Esta compatibilidad permite convertir **java.lang.Object** en **System.Object** y viceversa para asignaciones, paso de parámetros y otras tareas. Lo mismo ocurre con **java.lang.String** y **System.String**.

El consumo de archivos binarios de Visual J# no es diferente del consumo de cualquier otro archivo binario generado por lenguajes como Visual Basic .NET o Visual C#.

## Vea también

[Compatibilidad con el lenguaje](#)

# Extensiones de lenguaje para obtener compatibilidad con .NET Framework

Las siguientes características nuevas admiten el uso de .NET Framework:

- [ubyte](#)
- [Matrices rectangulares](#)
- [Nuevas directivas de preprocesamiento](#)
- [Asociar atributos](#)
- [Utilizar palabras clave como identificadores](#)
- [Llamar a métodos que toman argumentos por referencia](#)

## Vea también

[Sintaxis para utilizar .NET Framework](#)

# ubyte

**ubyte** es un nuevo tipo de datos primitivo de Visual J#. Se puede utilizar donde sea preciso un byte sin signo. Almacena valores en función del tamaño y el intervalo que se muestran en la siguiente tabla.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
ubyte	0 a 255	Entero de 8 bits sin signo	System.Byte

## Literales

Una conversión de asignación de un literal entero a un **ubyte** sólo se puede llevar a cabo si el valor del literal está dentro del intervalo de **ubyte** (0 a 255). Por ejemplo:

```
ubyte ub1 = 120; // OK within range
ubyte ub2 = 1200; // Compile time error. 1200 is not in the range of a ubyte
```

## Conversiones

Se puede llevar a cabo una conversión primitiva de ampliación de un **ubyte** a un **char**, **short**, **int**, **long**, **float** o **double**.

## Ejemplo

```
// vjc_ubyte.jsl
class MyClass
{
    public static void main(String [] args)
    {
        // An integral literal can be assigned to ubyte if it is in range
        ubyte ub1 = 250;
        System.Console.WriteLine(ub1); // Will print 250
        System.Console.WriteLine((byte)ub1); // -6
        // Create a Guid from a ubyte array
        ubyte val_array[] = new ubyte[] {0x57, 0x3e, 0x24, 0x76,
                                         0xef, 0x87, 0x4a, 0xed,
                                         0x81, 0x06, 0x59, 0x53,
                                         0xf3, 0xc1, 0x2b, 0x33};

        System.Guid g = new System.Guid(val_array);

        // Prints the Guid 76243e57-87ef-ed4a-8106-5953f3c12b33
        System.Console.WriteLine(g);
    }
}
```

## Salida

```
250
-6
76243e57-87ef-ed4a-8106-5953f3c12b33
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)

# Matrices rectangulares

Además de matrices escalonadas, Visual J# admite también matrices rectangulares, que siempre tienen una forma rectangular. El tipo de elemento y la forma de una matriz, incluido el número de dimensiones que tiene, forman parte de su tipo. Sin embargo, el tamaño de una matriz, representado por la longitud de sus dimensiones, no forma parte de su tipo.

Esta diferencia es más notable en la sintaxis del lenguaje Visual J#, puesto que la longitud de cada dimensión se especifica en la expresión de creación de la matriz y no en el tipo de la matriz. Por ejemplo, la declaración `int[, ] arr = new int[1, 1, 2];` es un tipo de matriz de `int[, ]` y una expresión de creación de matriz de `new int[1, 1, 2]`.

También es posible mezclar matrices rectangulares y escalonadas. Por ejemplo, `int [,][ ] mixedArr = new int[,][ ] { {{1,2}}, {{2,3,4}}};` es una matriz rectangular bidimensional mixta de matrices escalonadas. La matriz es rectangular de dimensiones `[2,1]`, donde cada elemento es una matriz de enteros unidimensional. La longitud de la matriz en `mixedArr[0, 0]` es 2, y en `mixedArr[1, 0]` es 3.

## Ejemplo

En los siguientes ejemplos se muestra cómo se pueden crear matrices rectangulares en Visual J#.

```
// vjc_rect_array.jsl
public class MyClass
{
    public static void main(String [] args)
    {
        // 2-D rectangular array; explicit creation.
        int[,] arr = new int[2, 2];
        arr[0, 0] = 1;
        arr[0, 1] = 2;
        arr[1, 0] = 3;
        arr[1, 1] = 4;

        // 2-D rectangular array;
        // Explicit creation with initializer list.
        int[,] arr2 = new int[,] {{1, 2}, {3, 4}};

        // 2-D rectangular array;
        // Shorthand: creation with initializer list.
        int[,] arr3 = {{1, 2}, {3, 4}};

        // 3-D rectangular array; explicit creation.
        int[, ,] arr4 = new int[1, 1, 2];
        arr4[0, 0, 0] = 1;
        arr4[0, 0, 1] = 2;

        // 3-D rectangular array;
        // Explicit creation with initializer list.
        int[, ,] arr5 = new int [, ,]
            {{{1,2,3},{1,2,3}},{1,2,3},{1,2,3}};

        // Mix rectangle & jagged arrays; explicit creation using
        // new. Creates the same array as the above example.
        int [,][ ] mixedArr = new int[,][ ] {{{1,2}}, {{2,3,4}}};

        // Mixed array dynamic creation
        int [,][ ] mixedArr2 = new int[2,1][ ];
        mixedArr2[0,0] = new int[] {1,2};
        mixedArr2[1,0] = new int[] {2,3,4};
    }
}
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)

# Nuevas directivas de preprocesamiento

Las siguientes directivas de procesamiento están disponibles ahora:

## [#line](#)

Permite modificar los números de línea que aparecen en los resultados de errores y advertencias del compilador.

## [#region](#)

Permite especificar un bloque de código que se puede expandir o contraer cuando se utiliza la característica de esquematización del editor de código de Visual Studio.

## [#endregion](#)

Marca el final de un bloque **#region**.

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)



# #line

**#line** permite modificar los números de línea y (opcionalmente) el nombre de archivo que aparecen en los resultados de errores y advertencias del compilador.

```
#line number ["file_name"]
#line default
```

donde:

*number*

Número que se desea especificar para la siguiente línea de un archivo de código fuente.

*file\_name* (opcional)

Nombre de archivo que debe aparecer en los resultados del compilador. El nombre predeterminado es el nombre real del archivo de código fuente. El nombre de archivo debe aparecer entre comillas dobles (""). Hay que utilizar el carácter delimitador '\' para delimitar un carácter '\' real cuando se especifique la ruta de un archivo de igual modo que para literales de cadena.

La directiva **#line default** define el número de línea con el valor original.

## Comentarios

**#line** podría utilizarse en un paso intermedio automatizado del proceso de generación. Por ejemplo, si el paso intermedio eliminó líneas del archivo de código fuente original, pero aún se desea que el compilador genere unos resultados basados en la numeración de líneas original del archivo, se pueden eliminar las líneas y, a continuación, simular la numeración original mediante **#line**.

Un archivo de código fuente puede tener cualquier número de directivas **#line**.

## Ejemplo

```
// preprocessor_line.js1
// compile with: /W:3
public class MyClass2
{
    public static void main(String [] args)
    {
        #line 200
        int i;    // VJS1493 on line 200
        #line 8 "c:\\hashline\\another_file.js1"
        char c;   // VJS1493 on line 8 of c:\\hashline\\another_file.js1
        #line default
        int r;    // VJS1493 on the original source line and file.
    }
}
```

## Vea también

[Nuevas directivas de preprocesamiento](#)

# #region

**#region** permite especificar un bloque de código que se puede expandir o contraer cuando se utiliza la función de esquematización del editor de código de Visual Studio.

```
#region name
```

donde:

*name*

Nombre que se desea asignar a la región y que aparecerá en el editor de código de Visual Studio.

## Comentarios

Un bloque **#region** debe terminarse con una directiva **#endregion**.

Un bloque **#region** no se puede solapar con un bloque **#if**. Sin embargo, un bloque **#region** puede estar anidado en un bloque **#if** y un bloque **#if** puede estar anidado en un bloque **#region**.

El compilador de Visual J# trata las líneas que comienzan con **#region** o **#endregion** como un comentario, por lo que estas directivas se omitirán en compilaciones desde la línea de comandos.

## Ejemplo

```
// preprocessor_region.jsl
#region MyClass definition
public class MyClass
{
    public static void main(String [] args)
    {
    }
}
#endregion
```

## Vea también

[Nuevas directivas de preprocesamiento](#)

# #endregion

**#endregion** marca el final de un bloque **#region**.

```
#endregion
```

Vea [#region](#) para obtener una descripción y un ejemplo de cómo utilizar **#endregion**.

## Vea también

[Nuevas directivas de preprocesamiento](#)

# Asociar atributos

Visual J# permite asociar atributos a clases, campos, métodos, parámetros y otros elementos de programación. Tanto los atributos personalizados como los pseudoatributos (atributos que no se pueden consultar mediante reflexión) se pueden asociar a los metadatos. Visual J# sigue la sintaxis de extensión de Visual J++ para permitir la asociación de atributos a los metadatos. Para asociar un atributo, utilice la directiva **@attribute** seguida del nombre del atributo que se va a asociar.

Los atributos se asocian a tipos de valor devuelto utilizando la directiva **@attribute.return** y se pueden orientar al método subyacente utilizando la directiva **@attribute.method**.

Es un error asociar estas directivas con elementos de programación que no sean los especificados. Para los demás atributos, el compilador determina el destino por medio del miembro siguiendo la directiva.

## Ejemplo

En el ejemplo siguiente, se muestra cómo utilizar **@attribute**.

```
// vjc_attributes1.jsl
// compile with: /target:library
import System.*;
import System.Runtime.InteropServices.*;
public class MyClass
{
    /** @attribute DllImport("user32", CharSet=CharSet.Ansi) */
    public static native int MessageBox(
        int hwnd,
        /** @attribute MarshalAs(UnmanagedType.AnsiBStr)*/
        System.String title,
        System.String caption,
        int type);
}
```

En el siguiente ejemplo se muestra el uso de **@attribute.return** para especificar el cálculo de referencias de atributos en un tipo de valor devuelto.

```
// vjc_attributes2.jsl
// compile with: /target:library
import System.Runtime.InteropServices.*;
public class MyClass
{
    /** @attribute.return MarshalAs(UnmanagedType.LPArray, SizeConst=10) */
    int[] getBytes()
    {
        int arr[] = {1,2};
        return arr;
    }
}
```

En el siguiente ejemplo se muestra el uso de **@attribute.method** para especificar el cálculo de referencias de atributos en un tipo de valor devuelto.

```
// vjc_attributes3.jsl
// compile with: /target:library
import System.*;
public class MyClass
{
    /** @property */
    /** @attribute.method Obsolete("get_Bytes is deprecated", false) */
    // The directive ensures that the attribute is attached to the method.
    int[] get_Bytes()
    {
        int arr[] = {1,2};
        return arr;
    }
}
```

```
}
```

No se puede definir un atributo en su programa de Visual J#; sólo se pueden utilizar atributos de .NET Framework u otros ensamblados a los que se hace referencia.

Los atributos de nivel de ensamblado se asocian utilizando la directiva **@assembly** en lugar de **@attribute**. La directiva **@assembly** debe estar al principio del archivo después de las instrucciones package e import, si las hay.

```
import System.Reflection.*;
/** @assembly AssemblyTitle("My Assembly") */
/** @assembly AssemblyCompany("My Company") */
```

## Gramática de los atributos

La gramática para los atributos es la siguiente:

*attributes:*

*attribute-sections*

*attribute-sections:*

*attribute-section*

*attribute-sections attribute-section*

*attribute-section:*

*/\*\* attribute-list \*/*

*attribute-list:*

*newline<sub>opt</sub> attribute newline<sub>opt</sub>*

*attribute-list newline attribute*

*attribute:*

*attribute-target-specifier attribute-name attribute-arguments*

*attribute-target-specifier:*

**@attribute**

**@assembly**

**@attribute.return**

**@attribute.method**

*attribute-name:*

*type-name*

*attribute-arguments:*

*( positional-argument-list<sub>opt</sub> )*

*( positional-argument-list , named-argument-list )*

*( named-argument-list )*

*positional-argument-list:*

*positional-argument*

*positional-argument-list , positional-argument*

*positional-argument:*

*attribute-argument-expression*

*named-argument-list:*

*named-argument*

*named-argument-list , named-argument*

*named-argument:*

*identifier = attribute-argument-expression*

*attribute-argument-expression:*

*literal-expression*

## Reglas para asociar atributos

- Un asterisco (\*) se trata como un espacio en blanco en la lista después del inicio del comentario. De manera similar, un carácter de línea nueva (\n) se trata como un espacio en blanco entre las palabras clave o los argumentos.
- Los argumentos se separan con comas.
- Los atributos deben comenzar al principio de una línea.
- Los atributos se pueden asociar a clases, campos, métodos o parámetros de método.
- El nombre de tipo que sigue a *@attribute-target-specifier* debe dar lugar a un tipo de atributo en el contexto actual. Se puede dar un nombre de clase completo o un nombre de clase sencillo, en cuyo caso la información de importación se utiliza para resolver el tipo. De manera similar, el identificador (*identifier*) del argumento con nombre debe corresponder a una propiedad o campo del tipo de atributo.
- Por convención, las clases de atributo se denominan con el sufijo Attribute. Un nombre de atributo de la forma nombre de tipo puede incluir u omitir este sufijo.

```
/** @attribute STAThreadAttribute() */ // refers to STAThreadAttribute
public static void main(String [] args) {}
// The Attribute suffix is optional
/** @attribute STAThread() */ // Also refers to STAThreadAttribute
public static void main(String [] args) {}
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)

# Utilizar palabras clave como identificadores

Visual J# permite tratar *@keyword* como un identificador para permitir el uso de identificadores en ensamblados a los que se hace referencia que son palabras clave reservadas de Visual J#.

## Ejemplo

```
// vjc_identifier_1.cs
// compile with: /target:library
// C# program
public class MyClass1
{
    public static int synchronized = 0;    // valid in C#
}
```

```
// vjc_identifier_2.jsl
// compile with: /reference:vjc_identifier_1.dll
public class MyClass2
{
    public static void main(String [] args)
    {
        MyClass1 x = new MyClass1();
        // System.Console.WriteLine(x.synchronized);    // error
        System.Console.WriteLine(x.@synchronized);    // OK
    }
}
```

## Salida

```
0
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)

# Llamar a métodos que toman argumentos por referencia

.NET Framework permite pasar parámetros por referencia. En este caso, el parámetro se puede modificar dentro de la función y estos cambios son visibles al llamador. Visual J# sigue el estilo de C++, donde se puede llamar directamente al método y pasar el parámetro como en el caso normal, y el compilador de Visual J# realiza la conversión implícitamente.

## Ejemplo

```
// vjc_args_by_ref_1.cs
// compile with: /target:library
// a C# program
public class MyClass1
{
    public void Test(ref string str)
    {
        str += " World!";
    }
}
```

Se puede llamar a este método en Java del siguiente modo:

```
// vjc_args_by_ref_2.jsl
// compile with: /reference:vjc_args_by_ref_1.dll
public class MyClass2
{
    public static void main(String [] args)
    {
        MyClass1 x = new MyClass1();
        System.String mystr = "Hello";
        System.Console.WriteLine(mystr);
        x.Test(mystr);
        System.Console.WriteLine(mystr);
    }
}
```

Sin embargo, Visual J# no admite la definición de métodos que tomen argumentos por referencia.

## Salida

```
Hello
Hello World!
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Extensiones de lenguaje para obtener compatibilidad con .NET Framework](#)



# Definir y utilizar propiedades

Visual J# permite tener acceso a propiedades definidas en clases de .NET Framework. Traduce las propiedades de las clases de .NET Framework a los métodos de descriptor de acceso **get\_** y **set\_**. Los desarrolladores de Visual J# sólo tienen que llamar a los métodos de descriptor de acceso directamente. Visual J# no asocia ninguna importancia especial a los descriptores de acceso de propiedad. Se tratan como métodos de la clase.

```
// vjc_properties.jsl
// compile with: /reference:System.Windows.Forms.dll
import System.Windows.Forms.*;
public class MyClass
{
    public static void main(String [] args)
    {
        Button b = new Button();
        String str = b.get_Text();
        System.Console.WriteLine("Button Text = " + str);
        b.set_Text("Sample Button");
        str = b.get_Text();
        System.Console.WriteLine("Button Text = " + str);
    }
}
```

También se pueden definir propiedades en Visual J#.

Cuando se definen métodos de descriptor de acceso de una propiedad, el nombre del método del descriptor de acceso debe comenzar con **get\_** o **set\_**. Los métodos de descriptor de acceso de una propiedad deben utilizar los mismos modificadores de acceso. El tipo de la propiedad debe ser el tipo de valor devuelto del método del descriptor de acceso **get\_** y el tipo del último argumento del método del descriptor de acceso **set\_**.

Se pueden reemplazar métodos de descriptor de acceso de propiedad en una clase derivada.

En Visual J#, una propiedad sencilla se define del siguiente modo:

```
// vjc_properties2.jsl
public class MyProp
{
    private int i = -1;

    /** @property */
    public int get_SomeNum()
    {
        return i;
    }

    /** @property */
    public void set_SomeNum(int input)
    {
        i = input;
    }

    public static void main()
    {
        MyProp aprop = new MyProp();
        System.Console.WriteLine(aprop.get_SomeNum());
        aprop.set_SomeNum(8);
        System.Console.WriteLine(aprop.get_SomeNum());
    }
}
```

## Salida

Las propiedades se pueden definir como de sólo lectura o de sólo escritura cuando sólo se ha definido la función de descriptor de acceso (**get\_** o **set\_**) correspondiente. También se pueden sobrecargar si los descriptors de acceso del método no crean ningún conflicto con las normas del lenguaje.

En el siguiente ejemplo se crean dos propiedades con el mismo nombre:

```
// vjc_properties3.js1
public class PropTest
{
    private int i = -1;
    private int j = -2;

    /** @property */
    public int get_SomeNum()
    {
        return i;
    }

    /** @property */
    public void set_SomeNum(int input)
    {
        i = input;
    }

    /** @property */
    public int get_SomeNum(int i)
    {
        return j;
    }

    /** @property */
    public void set_SomeNum(int i, int input)
    {
        j = input;
    }

    public static void main(String [] args)
    {
        int i = 0;
        PropTest aprop = new PropTest();
        System.Console.WriteLine(aprop.get_SomeNum());
        System.Console.WriteLine(aprop.get_SomeNum(i));
        aprop.set_SomeNum(8);
        aprop.set_SomeNum(i,9);
        System.Console.WriteLine(aprop.get_SomeNum());
        System.Console.WriteLine(aprop.get_SomeNum(i));
    }
}
```

### Salida

```
-1
-2
8
9
```

En el siguiente ejemplo se muestra un componente de Visual J# con una propiedad y se muestra cómo el cliente de Visual J# en primer lugar y un cliente de C# después utilizan la propiedad.

```
// vjc_prop_use.js1
// compile with: /target:library
public class PropTest
```

```

{
    private int i = -1;

    /** @property */
    public int get_SomeNum()
    {
        return i;
    }

    /** @property */
    public void set_SomeNum(int input)
    {
        i = input;
    }
}

```

En el siguiente ejemplo se muestra un cliente de Visual J# que utiliza las propiedades.

```

// vjc_prop_use2.jsl
// compile with: /reference:vjc_prop_use.dll
public class Test
{
    public static void main(String [] args)
    {
        int i = 0;
        PropTest apro = new PropTest();
        System.Console.WriteLine(aprop.get_SomeNum());
        apro.set_SomeNum(8);
        System.Console.WriteLine(aprop.get_SomeNum());
    }
}

```

#### Salida

```

-1
8

```

En el siguiente ejemplo se muestra un cliente de Visual C# que utiliza las propiedades.

```

// vjc_prop_use3.cs
// compile with: /reference:vjc_prop_use.dll
public class Test
{
    public static void Main()
    {
        PropTest apro = new PropTest();
        System.Console.WriteLine(aprop.SomeNum);
        apro.SomeNum = 8;
        System.Console.WriteLine(aprop.SomeNum);
    }
}

```

#### Salida

```

-1
8

```

El siguiente es un ejemplo de un componente de Visual J# que define una propiedad indizada. El siguiente ejemplo de Visual C# muestra cómo consumir la propiedad.

```

// vjc_indexed_properties.jsl

```

```

// compile with: /target:library
import System.*;
import System.Reflection.*;
class Employee
{
    public Employee(System.String s, int d)
    {
        _name = s;
        _dept = d;
    }

    /** @property */
    System.String get_name()
    {
        return _name;
    }

    /** @property */
    int get_dept()
    {
        return _dept;
    }

    private System.String _name;
    private int _dept;
}

/** @attribute DefaultMember("Report") */
// Sets the Report property as the indexer for the class.
class Manager
{
    /** @property */
    public Employee get_Report(System.String s)
    {
        for (pEmp = Reports ; (pEmp!=null) && (pEmp.emp.get_name() != s)
            ;
            pEmp = pEmp.next);
        if (pEmp!=null)
            return pEmp.emp;
        else
            return null;
    }

    /** @property */
    public void set_Report(System.String s, Employee e)
    {
        for (pEmp = Reports ; (pEmp!=null) && (pEmp.emp.get_name() != s)
            ;
            pEmp = pEmp.next);
        if (pEmp==null)
        {
            EmpList emp1 = new EmpList();
            emp1.emp = e;
            emp1.next = Reports;
            Reports = emp1;
        }
    }

    private static class EmpList    {
        public Employee emp;
        public EmpList next;
    }

    EmpList pEmp;
    static EmpList Reports = null;
}

```

En el siguiente ejemplo de Visual C#, se utiliza la propiedad definida en el ejemplo anterior:

```
// vjc_indexed_properties2.cs
// compile with: /reference:vjc_indexed_properties.dll
using System;

public class Class1
{
    public static void Main()
    {
        Manager Ed = new Manager();
        Employee Bob = new Employee("Bob Smith", 12);

        // track Ed's reports
        Ed[Bob.name] = Bob; // indexed by string type
        Console.WriteLine(Ed[Bob.name].dept);
    }
}
```

## Salida

12

Los atributos agregados a los métodos del descriptor de acceso se aplican a la propiedad de manera predeterminada. Sin embargo, puede utilizar la directiva **@attribute.method** para asociar los atributos con los métodos individuales, como se muestra a continuación:

```
// vjc_properties_and_attributes.jsl
// compile with: /target:library
import System.ComponentModel.*;

/*
 * Different attributes to getter and setter methods
 */
public class Scalar
{
    private int magnitude = 101;

    /**@attribute Description("Attribute on the property")*/
    /**@attribute.method Description("Attribute on get accessor")*/
    /**@property*/
    public int get_Magnitude()
    {
        return magnitude;
    }

    /**@attribute.method Description("Attrib on set accessor")*/
    /** @property */
    public void set_Magnitude(int value)
    {
        magnitude = value;
    }
}
```

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Ejemplo: propiedades y eventos](#)

# Definir y utilizar eventos

Visual J# permite registrar eventos publicados por clases de .NET Framework. No aporta especial importancia a los eventos y se pueden registrar controladores de evento utilizando los descriptores de acceso de evento **add\_** y **remove\_**, como se muestra a continuación.

```
// vjc_events.jsl
// compile with: /reference:System.Windows.Forms.dll
import System.Windows.Forms.*;

public class MyClass
{
    public static void main(String [] args)
    {
        System.Windows.Forms.Button button = new System.Windows.Forms.Button();
        MyClass myClass = new MyClass();
        System.EventHandler eventhandler = new System.EventHandler(myClass.OnDoubleClick);
        button.add_DoubleClick(eventhandler);
        // where OnDoubleClick is a method defined in the current class
        // that has the same signature of the EventHandler delegate
        // The code above demonstrates how to use .NET delegates in Visual J#

        // similarly, you can unsubscribe to the event as follows
        button.remove_DoubleClick(eventhandler);
    }

    private void OnDoubleClick(System.Object sender, System.EventArgs e)
    {
    }
}
```

En Visual J# se pueden definir eventos de .NET Framework.

El formato de una declaración de evento es:

```
/** @event */
void add_EventName(event_type event)

/** @event */
void remove_EventName(event_type event)
```

donde `event_type` será un parámetro cuyo tipo define el tipo del evento y se debe derivar de **System.Delegate**. Los métodos **add** y **remove** de un evento deben estar los dos presentes o los dos ausentes y tener los mismos modificadores de acceso.

Se pueden reemplazar métodos de descriptor de acceso de evento en una clase derivada, como se muestra a continuación:

```
// vjc_events2.jsl
import System.*;
import System.Collections.ArrayList;

/** @delegate */
public delegate void MyDelegate(); // delegate declaration

public interface I
{
    /** @event */
    public void add_MyEvent(MyDelegate listener);

    /** @event */
    public void remove_MyEvent(MyDelegate listener);

    void FireAway();
}
```

```

public class MyClass implements I
{
    private ArrayList list = new ArrayList(10);
    private int no = 0;

    /** @event */
    public void add_MyEvent(MyDelegate listener)
    {
        list.Add(listener);
    }

    /** @event */
    public void remove_MyEvent(MyDelegate listener)
    {
        list.Remove(listener);
    }

    public void FireAway()
    {
        System.Object [] toArray = list.ToArray();

        int len = toArray.length;
        for (int i = 0; i < len ; i++)
        {
            ((MyDelegate)(toArray[i])).Invoke();
        }
    }
}

public class MainClass
{
    static public void main (String [] args)
    {
        new MainClass();
    }

    public void f1()
    {
        Console.WriteLine("This is called when the event fires.");
    }

    public MainClass()
    {
        I i = new MyClass();

        i.add_MyEvent(new MyDelegate(this.f1));
        i.FireAway();
    }
}

```

## Salida

```
This is called when the event fires.
```

Si el programa anterior se compilara como .DLL, se podría tener acceso a su evento desde un programa de Visual C# del siguiente modo:

```

// vjc_events3.jsl
// compile with: /target:library
import System.*;
import System.Collections.ArrayList;

/** @delegate */
public delegate void MyDelegate(); // delegate declaration

```

```

public interface I
{
    /** @event */
    public void add_MyEvent(MyDelegate listener);

    /** @event */
    public void remove_MyEvent(MyDelegate listener);

    void FireAway();
}

public class MyClass implements I
{
    private ArrayList list = new ArrayList(10);
    private int no = 0;

    /** @event */
    public void add_MyEvent(MyDelegate listener)
    {
        list.Add(listener);
    }

    /** @event */
    public void remove_MyEvent(MyDelegate listener)
    {
        list.Remove(listener);
    }

    public void FireAway()
    {
        System.Object [] toArray = list.ToArray();

        int len = toArray.length;
        for (int i = 0; i < len ; i++)
        {
            ((MyDelegate)(toArray[i])).Invoke();
        }
    }
}

```

Cliente de Visual C#:

```

// vjc_events4.cs
// compile with: /reference:vjc_events3.dll
using System;
public class MainClass
{
    static private void f()
    {
        Console.WriteLine("This is called when the event fires.");
    }

    static public void Main ()
    {
        I i = new MyClass();

        i.MyEvent += new MyDelegate(f);
        i.FireAway();
    }
}

```

**Salida**

This is called when the event fires.



**Vea también**

[Sintaxis para utilizar .NET Framework | Ejemplo: propiedades y eventos](#)

# Definir y utilizar delegados de .NET Framework

Los delegados de .NET Framework se tratan como objetos de primera clase y el compilador de Visual J# permite crear un delegado de .NET Framework, igual que la extensión de delegados de Visual J++ . Una diferencia importante consiste en que los enlaces se realizan siempre en tiempo de compilación.

```
// vjc_delegates.jsl
import System.Windows.Forms.*;
import System.*;

public class MyObjClass
{
    public void OnDoubleClick(System.Object o, System.EventArgs e)
    {
        System.Console.WriteLine("test");
    }
}

public class MyClass
{
    public static void main()
    {
        MyObjClass obj = new MyObjClass();

        // Where OnDoubleClick is a method in obj.
        EventHandler handler = new EventHandler(obj.OnDoubleClick);
        handler.Invoke(obj, new EventArgs());
        // The following is also valid.
        EventHandler handler2 = new EventHandler(obj, "OnDoubleClick");
        handler2.Invoke(obj, new EventArgs());
    }
}
```

## Salida

```
test
test
```

En Visual J# se pueden definir delegados de .NET Framework. También se pueden definir delegados al estilo de Visual J++ 6.0 y son de tipo **com.ms.lang.Delegate**.

La declaración de un tipo de delegado constará de los siguientes componentes:

- Atributo `/** @delegate */`
- Nivel de acceso
- Delegado de palabra clave (o delegado de multidifusión)
- Tipo de valor devuelto y firma del método que controla el tipo de delegado
- Nombre del tipo de delegado (que se coloca entre el tipo de valor devuelto y la firma del método)

En el siguiente código se declara **EventHandler** para que sea un tipo de delegado público que controla métodos que no toman parámetros y con un tipo de valor devuelto de void:

```
/** @delegate */
public delegate void EventHandler();
```

Si se utiliza un tipo de delegado para controlar un único método sencillo una vez, se puede controlar una función miembro de cualquier tipo de valor devuelto y firma. Si, no obstante, el tipo de delegado controla dos o más métodos simultáneamente, el tipo de valor devuelto debe ser void.

El enlace de delegados .NET se realiza siempre en tiempo de compilación.

```
// vjc_delegates2.jsl
import System.*;

/**@delegate*/
delegate void MyDelegate(int i);

class Program
{
    public static void main(System.String [] args)
    {
        TakesADelegate(new MyDelegate(Program.DelegateFunction));
    }

    public static void TakesADelegate(MyDelegate SomeFunction)
    {
        SomeFunction.Invoke(21);
    }

    public static void DelegateFunction(int i)
    {
        Console.WriteLine("Called by delegate with number: " + i + ".");
    }
}
```

### Salida

```
Called by delegate with number: 21.
```

### Vea también

[Sintaxis para utilizar .NET Framework](#) | [Ejemplo: propiedades y eventos](#)

# Utilizar tipos de valor

El compilador de Visual J# permite utilizar tipos de valor definidos en .NET Framework u otros ensamblados definidos por el usuario. Aunque Java no permite definir tipos de valor, se pueden utilizar instancias de tipos de valor directamente en Visual J# como **System.Object**. El compilador de Visual J# realiza la conversión boxing necesaria de manera implícita. De forma similar, cuando se convierte un objeto en un tipo de valor, el compilador lleva a cabo la conversión unboxing.

## Ejemplo

```
// vjc_valuetypes1.jsl
import System.*;
public class MyClass
{
    public static void main(String [] args)
    {
        // DateTime is a value type; use it like a reference type
        DateTime dt = new DateTime();
        dt = DateTime.Parse("01/01/2002 12:00");

        // automatically box value type dt to System.Object
        System.Object obj = dt;

        // obj unboxed to the value type DateTime
        DateTime dt2 = (DateTime) obj;
        dt2 = DateTime.Parse("01/01/2003 12:00");
    }
}
```

En este código, `DateTime` es un tipo de valor definido en la biblioteca de .NET Framework. Aunque Java no tiene el concepto de tipos de valor, Visual J# hace posible que los desarrolladores de Java utilicen tipos de valor definidos en la biblioteca de .NET Framework. Sin embargo, no se puede definir un tipo de valor en Visual J#; sólo se pueden utilizar tipos de valor de otros ensamblados a los que se hace referencia.

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Tipos de valor](#) | [Ejemplo: propiedades y eventos](#)

# Utilizar tipos de valor correspondientes a tipos primitivos

Visual J# permite utilizar directamente los tipos de clase primitivos correspondientes a cada tipo primitivo que se pueda encontrar en .NET Framework (**System.Int32** para **int**). La única limitación es que se debe utilizar una conversión para asignar un tipo primitivo a su tipo de clase correspondiente. De manera similar, se necesita una conversión para asignar un tipo primitivo a **System.Object**. Estas limitaciones son necesarias para no infringir la semántica de Java. La conversión del tipo primitivo en el tipo de valor correspondiente no es transitiva.

## Ejemplo

```
// vjc_valuetypes2.jsl
import System.*;
public class MyClass
{
    public static void main(String [] args)
    {
        Int32 int32 = (Int32) 10;
        int i = (int) int32;
        Console.WriteLine(i);
        System.Object obj1 = int32;    // will work as int32 is a value type
        Console.WriteLine(obj1);
        System.Object obj2 = (Int32) 10;    // will also work
        Console.WriteLine(obj2);

        // The following statements will not compile,
        // a primitive type cannot be assigned to a reference type
        // System.Int32 int32 = 10;
        // System.Object obj = 10;
    }
}
```

## Salida

```
10
10
10
```

.NET Framework admite algunos tipos primitivos que no forman parte de Java. Se pueden utilizar estos tipos usando el tipo de valor correspondiente de .NET Framework como se explicó antes, y el compilador de Visual J# genera el código correcto para las conversiones. En la siguiente tabla se ofrece una lista de estos tipos:

Tipo primitivo de .NET Framework no compatible con Visual J#	Tipo de valor correspondiente en .NET Framework
unsigned short o uint16	System.UInt16
unsigned int o uint32	System.UInt32
unsigned long o uint64	System.UInt64
native int	System.IntPtr
native unsigned int	System.UIntPtr

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Ejemplo: propiedades y eventos](#)

# Utilizar enumeraciones

El compilador de Visual J# permite utilizar enumeraciones definidas en .NET Framework u otros ensamblados definidos por el usuario. Las enumeraciones se tratan como tipos de referencia de forma predeterminada. Las enumeraciones se pueden convertir en sus tipos primitivos subyacentes y viceversa. También se pueden asignar a **System.Object**, en cuyo caso el compilador lleva a cabo la conversión boxing necesaria de manera implícita.

## Ejemplo

```
// vjc_enums.jsl
import System.*; // DayOfWeek enumerator defined in .NET Framework
public class MyClass
{
    public static void main()
    {
        DayOfWeek friday = DayOfWeek.Friday;

        // Call a method that has a parameter of type DayOfWeek
        TestClass mySchedule = new TestClass();
        mySchedule.SetNonWorkingDay(friday);
        int i = (int) friday; // convert an enum to its underlying primitive
        Console.WriteLine(i);

        // OK to cast a primitive type to enum
        DayOfWeek monday = (DayOfWeek) 1;
        Console.WriteLine(monday);

        // following line automatically boxes enum friday to System.Object
        System.Object obj = friday;
        Console.WriteLine(obj);
    }
}

public class TestClass
{
    public void SetNonWorkingDay(DayOfWeek dow)
    {
    }
}
```

## Salida

```
5
Monday
Friday
```

Se puede aplicar operadores bit a bit (|, &, ^) a tipos de enumeración. El código siguiente lleva a cabo una operación OR bit a bit en `System.Windows.Forms.AnchorStyles` en una llamada a `CheckBox.set_Anchor`:

```
CheckBox.set_Anchor(AnchorStyles.Top | AnchorStyles.Bottom)
```

No se pueden definir enumeraciones en Visual J#.

## Vea también

[Sintaxis para utilizar .NET Framework](#) | [Ejemplo: propiedades y eventos](#)

# Ejemplo: propiedades y eventos

En el siguiente ejemplo se muestra un componente de Visual J# que expone propiedades y eventos:

```
// vjc_container.jsl
// compile with: /target:library
import System.Collections.ArrayList;

/** @delegate */
public delegate void ValueChanged();

public class Container
{
    private ArrayList al = new ArrayList();

    // Readonly property Count
    /** @property */
    public int get_Count()
    {
        return al.get_Count();
    }

    /** @property */
    public void set_Value(int i, System.String val)
    {
        al.Insert(i, val);
        if (ev != null)
        {
            ev.Invoke(); // raise event
        }
    }

    /** @property */
    public System.String get_Value(int i)
    {
        return (System.String) al.get_Item(i);
    }

    public ValueChanged ev = null;

    /** @event */
    public void add_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Combine(ev, p);
    }

    /** @event */
    public void remove_Event(ValueChanged p)
    {
        ev = (ValueChanged) System.Delegate.Remove(ev, p);
    }
}
```

A continuación, se muestra un cliente de Visual Basic que utiliza las propiedades y recibe el evento desencadenado por el componente de Visual J#:

```
' client.vb
' compile with: /reference:vjc_container.dll
Imports System
Public Class Form
    WithEvents Dim c As New Container()

    Public Shared Sub Main()
        Dim f As New Form
    End Sub
```

```
Public Sub New()  
    c.Value(0) = "String1"  
    c.Value(1) = "String2"  
  
    ' Fetching read-only property  
    System.Console.WriteLine(c.Count)  
End Sub  
  
' Sink Visual J# event  
Public Sub EvListener() Handles c.Event  
    System.Console.WriteLine("Visual J# Event Fired")  
End Sub  
End Class
```

## **Vea también**

[Sintaxis para utilizar .NET Framework](#) | [Utilizar eventos](#) | [Definir y utilizar propiedades](#)



# Llamar a métodos nativos

Visual J# proporciona compatibilidad total con la tecnología J/Direct disponible en Visual J++ 6.0. Además, puede utilizar los servicios de invocación de plataforma que proporciona Common Language Runtime para llamar al código nativo. Para obtener más información, vea [Utilizar funciones de DLL no administradas](#).

## Ejemplo

En el siguiente código, se utiliza el mecanismo de invocación de plataforma para generar texto utilizando funciones de la DLL de tiempo de ejecución de Microsoft Visual C++.

```
// vjc_pinvoke.jsl
import System.Runtime.InteropServices.*;
class MyClass
{
    /** @attribute DllImport("msvcrt.dll") */
    public static native int puts(String c);
    /** @attribute DllImport("msvcrt.dll") */
    private static native int _flushall();

    public static void main(String [] args)
    {
        puts("Test");
        _flushall();
    }
}
```

Cuando se utiliza la invocación de plataforma para llamar a código nativo, sólo se permite el cálculo de referencias para:

- Tipos definidos en .NET Framework
- **java.lang.Object**
- **java.lang.String**
- Tipos primitivos

Para convertir otros tipos de Java en código nativo, puede ser necesario utilizar cálculo de referencias personalizado, puesto que estos tipos no se controlan con el cálculo de referencias predeterminado de la invocación de plataforma de .NET Framework. Para obtener más información, vea [Cálculo de referencias de interoperabilidad](#).

Al actualizar aplicaciones de Visual J++ 6.0, no se admite el uso de directivas **@dll** y **@com** de Visual J++ 6.0 con los atributos de interoperabilidad de .NET Framework, por lo que está totalmente desaconsejado. Esto podría producir problemas en la aplicación compilada en tiempo de ejecución.

## Salida

Test

## Vea también

[Sintaxis para utilizar .NET Framework](#)

# Operadores de conversión de tipos

Visual J# no admite operadores de conversión de tipos. Un tipo compatible con CLS (Common Language Specification) suele proporcionar un mecanismo alternativo para lograr la conversión utilizando los métodos **ToX** (donde **X** es el nombre del tipo destino) y **FromY** (donde **Y** es el nombre del tipo origen). El usuario de Visual J# puede llevar a cabo las conversiones de tipos utilizando estos métodos.

No se recomienda el uso directo de los métodos **op\_implicit** y **op\_explicit**, ya que estos métodos pueden tener tipos de valor devuelto reemplazados y, por tanto, no se pueden utilizar para la resolución de métodos.

## Ejemplo

```
// vjc_type_conv_op.jsl
import System.*;
class MyClass
{
    public static void main(String [] args)
    {
        int i = 10;
        Decimal dec = new Decimal(20);
        Console.WriteLine("dec = {0}", dec);
        Console.Write("i = ");
        Console.WriteLine(i);

        // In Visual J#, we can use the CLS-compliant conversion methods
        i = Decimal.ToInt32(dec);
        Console.Write("i = ");
        Console.WriteLine(i);

        // Converting an int to a System.Decimal
        // In Visual J#, we can use the constructor that takes int
        i = 9;
        dec = new System.Decimal(i);
        Console.WriteLine("dec = {0}", dec);
    }
}
```

## Salida

```
dec = 20
i = 10
i = 20
dec = 9
```

## Vea también

[Sintaxis para utilizar .NET Framework](#)

# Características de .NET Framework no compatibles

Las características siguientes, que pueden ser compatibles con otros lenguajes orientados a Common Language Runtime de .NET Framework, no son compatibles con Visual J#:

- Sobrecarga de operadores y semántica de .NET Framework asociada a ella.
- Conversiones implícitas y explícitas entre tipos que utilizan los operadores de conversión **op\_Implicit** y **op\_Explicit**.
- Compatibilidad para definir las siguientes construcciones de .NET Framework:
  - Tipos de valor
  - Atributos personalizados
  - Enumeraciones
- Conversión sin problemas entre tipos de datos de Java y tipos de datos de .NET Framework.
- Comprobación del compilador de compatibilidad con CLS ([CLSCompliantAttribute \(Clase\)](#))
- Si un tipo simple implementa dos interfaces y en ambas hay que definir un método con el mismo nombre y la misma firma, Visual J# no los considera métodos distintos con diferentes implementaciones. Visual J# sólo admite un único cuerpo de código para implementar todos los métodos de interfaces que tengan el mismo nombre y la misma firma.

## Vea también

[Compatibilidad con el lenguaje](#) | [Bibliotecas de clases y API no compatibles](#) | [Escenarios no compatibles con Visual J#](#) | [Problemas en la generación de un proxy para servicios Web](#)

# Generar y compilar código fuente dinámicamente en Java

Visual J# proporciona una implementación de CodeDOM para Java. La clase **Microsoft.VJSharp.VJSharpCodeProvider** del ensamblado **VJSharpCodeProvider.dll** proporciona métodos para recuperar instancias de las interfaces **ICodeGenerator** y **ICodeCompiler** implementadas en Visual J# para el lenguaje Java.

## Para incluir en un proyecto el proveedor de código de Visual J#

- Haga clic con el botón secundario del *mouse* (ratón) en el **Explorador de soluciones**, señale **Agregar referencia** y seleccione **VJSharpCodeProvider** en la lista de componentes .NET.

Los elementos de CodeDOM se pueden usar para crear código fuente de manera independiente del lenguaje. La implementación específica del lenguaje de **ICodeGenerator** e **ICodeCompiler** puede utilizarse para generar y compilar código de manera dinámica.

Para obtener más información acerca de CodeDOM, vea [Generar y compilar código fuente dinámicamente en varios lenguajes](#).

CodeDOM es compatible con los tipos comunes de elementos de código presentes en los lenguajes de programación usuales. Sin embargo, no está diseñado para proporcionar elementos que representen todas las características posibles de un lenguaje de programación.

A continuación se describen formas de proporcionar compatibilidad con características específicas de Visual J# (como la cláusula **throws**) y formas de diferenciar las cláusulas **extends** e **implements**.

**Nota** Si a la implementación de **ICodeGenerator** en otro lenguaje de programación se le pasa el mismo gráfico de objetos de CodeDOM, omitirá la información de **UserData** que no sea relevante para dicho lenguaje.

CodeDOM no proporciona ningún modo de representar las excepciones declaradas en la cláusula **throws** de un método.

## Para representar la cláusula throws

- Establezca un par clave-valor en el miembro **CodeObject.UserData** del objeto **CodeMemberMethod** que representa el método con la cláusula **throws**.

La implementación de **ICodeGenerator** en Visual J# interpreta este miembro de forma apropiada.

- Asigne a la clave el nombre **throwsCollection**. Su valor debería ser un objeto **CodeTypeReferenceCollection** que representa a una colección de objetos **CodeTypeReference** que describen las excepciones provocadas por el método.
- El generador de Visual J# utiliza **throwsCollection** para generar la cláusula **throws** para el método.

Para ver un ejemplo, vea [Ejemplo CodeDOM](#).

CodeDOM no proporciona una forma de representar la clase base de una clase de forma independiente de la interfaz base. De forma predeterminada, el generador de código de Visual J# interpreta **CodeTypeDeclaration.BaseTypes** de la forma siguiente:

- El primer tipo de la colección **BaseTypes** es la clase base de la clase (corresponde a la cláusula **extends**).
- Los tipos restantes de la colección **BaseTypes** son las interfaces implementadas por la clase (corresponde a la cláusula **implements**).

El generador de código de Visual J# requiere información adicional para generar una declaración de clase que no incluya explícitamente la cláusula **extends**.

## Para diferenciar las cláusulas extends e implements

- Establezca un par clave-valor en el miembro **UserData** del objeto **CodeTypeDeclaration** que representa la clase que va a generar.
- Asigne a la clave el nombre **hasExtendsClause**. Debería tener como valor un objeto **Boolean**.
- El generador de código interpreta el valor del siguiente modo:
  - Si **hasExtendsClause** tiene el valor **true**, el primer tipo de la colección **BaseTypes** es la clase base de la clase (corresponde a la cláusula **extends**). Los demás tipos de la colección **BaseTypes** son las interfaces implementadas por la clase (corresponde a la cláusula **implements**).
  - Si el valor de **hasExtendsClause** es **false**, todos los tipos de la colección **BaseTypes** se interpretan como interfaces implementadas por la clase.

Encontrará un ejemplo en [Ejemplo CodeDOM](#).

## **Vea también**

[Compatibilidad con el lenguaje](#) | [Generar y compilar código fuente dinámicamente en varios lenguajes](#) | [Problemas en la generación de un proxy para servicios Web](#) | [CodeDOM \(Ejemplo\)](#)

# Problemas en la generación de un proxy para servicios Web

En un proyecto de Visual J#, cuando se agrega una referencia a un servicio Web XML, se genera un proxy para el servicio Web y se agrega al proyecto. Este proxy se utiliza para invocar al método de servicios Web. El proveedor CodeDOM genera el código fuente para el proxy. Hay determinados casos en los que puede fallar la generación de un proxy debido a características del lenguaje que no están disponibles en Visual J#, como se muestra a continuación:

- El método de servicios Web tiene argumentos que se pasan por referencia.
- Los parámetros son de entrada y salida.
- Los parámetros o tipos de valor devuelto son enumeraciones.

En estos casos, el proveedor CodeDOM inserta instrucciones de error en el archivo de código fuente generado, de manera que se produce un error en tiempo de compilación en el proyecto.

Una alternativa consiste en generar el proxy en Visual Basic o Visual C# y utilizarlo en el proyecto de Visual J#. Puede hacer esto de varias maneras:

1. Agregue un proyecto de biblioteca de controles de Visual Basic o Visual C# a la solución.
2. Agregue la referencia Web al nuevo proyecto.
3. Agregue este proyecto como una referencia al proyecto de Visual J#.

O bien

1. Abra una línea de comandos de Visual Studio .NET.
2. Utilice la herramienta wsdl.exe para generar la clase de proxy para la referencia Web en Visual Basic o Visual C#.
3. Compile el archivo generado en una biblioteca (.dll) utilizando el compilador de línea de comandos apropiado para el lenguaje (csc.exe o vbc.exe).
4. Agregue la DLL al proyecto de Visual J# como una referencia.

**Nota** Para métodos de servicios Web con enumeraciones, Visual J# puede generar servidores proxy si se utiliza el estilo de documento/literal. Es decir, en la definición del servicio WSDL, si el atributo **style** del elemento `<soap:binding>` se define como **document** (y no como **rpc**) y el atributo **use** del elemento de enlace `<soap:body>` no se define como **Encoded**, no se codifican los mensajes SOAP que comunican con el servicio Web. En este escenario, se genera una clase de proxy con valores constantes y un campo de cadena para el tipo de enumeración. Sin embargo, si se codifican los mensajes SOAP, Visual J# no admite la generación de clases de proxy para este método de servicios Web

## Vea también

[Compatibilidad con el lenguaje](#) | [Generar y compilar código fuente dinámicamente en Java](#)

# Comentarios sobre documentación en Visual J#

En esta sección, se proporciona documentación de referencia sobre la compatibilidad con comentarios de documentación en el entorno de desarrollo de Visual J#.

## En esta sección

### [Compatibilidad con comentarios de documentación en el Editor de código de Visual J#](#)

Explica las características del Editor de código que admiten comentarios de documentación, con el formato conocido como comentarios Javadoc para el código fuente de Java.

### [Generar páginas Web de comentarios](#)

Explica el elemento de menú Generar páginas Web de comentarios, que se puede utilizar para generar documentación a partir de los archivos de código fuente.

## Secciones relacionadas

### [Compatibilidad con el lenguaje](#)

Proporciona información de referencia sobre las funciones de Visual J++<sup>®</sup> 6.0 y .NET Framework que son compatibles y no compatibles con Visual J#.

### [Ver la estructura de código con comentarios](#)

Explica cómo crear una serie de páginas HTML utilizadas para examinar la estructura y los comentarios de código incluidos en proyectos escritos en un lenguaje de programación que admita comentarios de documentación XML.

# Compatibilidad con comentarios de documentación en el Editor de código de Visual J#

Visual J# admite comentarios de documentación, con el formato conocido como comentarios Javadoc para el código fuente de Java.

## Autocompletar comentarios de documentación

En el Editor de código de Visual J#, al escribir los caracteres de apertura `/**` de un comentario Javadoc, el editor inserta automáticamente los caracteres de cierre `*/` en la línea siguiente.

## Autocompletar etiquetas en comentarios Javadoc

Las etiquetas Javadoc y las directivas de extensiones de Microsoft incluidas en Visual J+ + 6.0 comienzan con el carácter `@` en los comentarios Javadoc. En Visual J#, si se escribe el carácter `@` en un comentario Javadoc, se obtiene una lista de autocompletar que incluye las etiquetas Javadoc y las directivas de extensiones de Microsoft admitidas.

## IntelliSense en comentarios Javadoc

Los programas de Visual J# hacen un amplio uso de la sintaxis de declaración de atributos con el formato de comentario Javadoc. En un comentario Javadoc, cuando se escribe **@attribute**, aparece la lista desplegable IntelliSense, que incluye las clases accesibles.

## Vea también

[Comentarios sobre documentación en Visual J#](#)



# Generar páginas Web de comentarios

Utilice el comando **Generar páginas Web de comentarios** del menú **Herramientas** para generar documentación a partir de los comentarios Javadoc de los archivos de código fuente.

Para un elemento, se generan las secciones Resumen y Descripción a partir del comentario Javadoc asociado utilizando:

- La frase primera (o de resumen) del comentario Javadoc
- El resto del comentario Javadoc.

Para métodos y constructores, se utilizan las etiquetas Javadoc **@param** y **@return** para generar los comentarios de los parámetros y valores devueltos. Las etiquetas **@see**, **@version** y **@author** se omiten.

Los comentarios generados contienen también la clase base, el nivel de acceso de la clase o miembro, las firmas de las funciones y una lista de sobrecargas de funciones para cada método.

Cargue el ejemplo siguiente en un proyecto y seleccione **Generar páginas Web de comentarios** en el menú **Herramientas** para ver los resultados.

```
// vjc_build_comment_web_pages.jsl
// compile with: /target:library
/**
 * Class that represents a complex number.
 * This class is used to represent a complex number with an
 * imaginary and real part.
 *
 * @see      SimpleClass
 * @version  1.0
 * @author   tom
 */
public class ComplexClass
{
    /**
     * Stores the real part.
     */
    private double realpart = 0.0;

    /**
     * Stores the imaginary part.
     */
    private double imaginarypart = 0.0;

    /** Default constructor for ComplexClass.
     * <em>It is recommended to use the constructor
     * with two arguments.</em>
     */
    public ComplexClass()
    {
    }

    /** Constructor for ComplexClass.
     * @param   real      The value to set for the real part.
     * @param   imaginary  The value to set for the imaginary part.
     */
    public ComplexClass(double real, double imaginary)
    {
        realpart = real;
        imaginarypart = imaginary;
    }

    /**
     * Returns the real part of the number.
     * @return   The real part of the complex number.
     */
    public double getRealPart()
    {
        return realpart;
    }
}
```

```
}

/**
 * Returns the imaginary part of the number.
 * @return The imaginary part of the complex number.
 */
public double getImaginaryPart()
{
    return imaginarypart;
}
}
```

## **Vea también**

[Ver la estructura del código con comentarios](#) | [Comentarios sobre documentación en Visual J#](#) |  
[Generar páginas Web de comentarios \(Cuadro de diálogo\)](#)

# Jerarquías de excepciones de Visual J#

Los programadores de aplicaciones en Visual J# deben tratar con las dos jerarquías de excepciones siguientes:

- Excepciones de Java, derivadas de **java.lang.Throwable**. En ellas podemos distinguir dos categorías:

Excepción	Descripción
Checked	Si un método produce una excepción comprobada, el compilador genera un error, a menos que el código que llama al método disponga de un bloque <code>try-catch</code> para capturar la excepción comprobada.
Runtime	Derivada de <b>java.lang.RuntimeException</b> . El compilador no exige estas capturas de excepciones en el código de llamada.

- Las excepciones de .NET Framework se derivan de **System.Exception**. En estas excepciones no existen las categorías definidas previamente.

Muchas de las excepciones de tiempo de ejecución de Java son semánticamente equivalentes a las de .NET Framework; por ejemplo, **java.lang.NullPointerException** es similar a **System.NullReferenceException**. En

[Excepciones de Visual J# con equivalentes de .NET Framework](#) verá una tabla con las correspondencias entre las excepciones de tiempo de ejecución de Java y las de .NET Framework.

Al capturar en el código fuente una excepción de tiempo de ejecución de Java, el compilador de Visual J# se asegura de que el código ejecutable resultante también capture la excepción equivalente de .NET Framework. Por ejemplo:

```
try {
    methodWithNullReference();
} catch (java.lang.NullPointerException e) {
    // Catches both java.lang.NullPointerException
    // and System.NullReferenceException.
    e.printStackTrace();
}
```

En este ejemplo hay dos formas posibles de generar una excepción de referencia null:

- Se desencadena explícitamente en el método `methodWithNullReference` como excepción de tipo **NullPointerException**.

O bien

- Se produce en tiempo de ejecución como una excepción de tipo **NullReferenceException** de .NET Framework.

La cláusula `catch (java.lang.NullPointerException e)` capturará tanto las excepciones de tipo **NullPointerException** como las de tipo **NullReferenceException**.

De esta forma, puede centrarse en las excepciones específicas de Java y, en general, podrá pasar por alto las de .NET Framework que tengan equivalentes en Java. Sin embargo, en la cláusula `catch` deben especificarse las excepciones de .NET Framework que no aparezcan en la tabla.

En Visual J#, en lugar de capturar **Throwable** en todas las excepciones (como en Java) se captura **System.Exception**, de la que se deriva **Throwable**. Por ejemplo:

```
try {
    ...
} catch (System.Exception e) {
    // catches all exceptions
    e.printStackTrace();
}
```

**Nota** En general, no es necesario capturar todas las excepciones. Para hacerlo tendría que crear código específico para cada excepción que pueda producirse.

## Vea también

[Compatibilidad con el lenguaje](#) | [Excepciones de Visual J# con equivalentes de .NET Framework](#)

# Compatibilidad con bibliotecas de clases

Visual J# proporciona un conjunto de bibliotecas de clases desarrolladas de manera independiente y diseñadas para proporcionar la funcionalidad de la mayoría de las bibliotecas de clases de JDK nivel 1.1.4 e incluye clases especificadas en el currículo de becas avanzadas de informática (Advanced Placement curriculum for Computer Science) del sistema College Board en Estados Unidos. Visual J# admite también las extensiones de Microsoft® que se distribuyen con Microsoft Visual J++® 6.0, incluidas las clases WFC (Windows Foundation Classes) y muchas otras bibliotecas de clases com.ms.\*.

En esta sección se enumeran los paquetes compatibles y no compatibles con Visual J#, y se proporciona información detallada específica de la implementación de la compatibilidad con bibliotecas de clases.

## En esta sección

### [Bibliotecas de clases compatibles](#)

Enumera el conjunto de bibliotecas de clases disponibles en Visual J#.

### [Bibliotecas de clases y características no compatibles](#)

Enumera las bibliotecas de clases y API no compatibles con Visual J#.

### [Métodos de la clase java.lang.Class](#)

Proporciona información detallada sobre los métodos **Class.FromType** y **Class.ToType** de **java.lang.Class**.

### [Métodos de la clase com.ms.wfc.app.Locale](#)

Muestra los métodos no admitidos de la clase **com.mswfc.app.Locale**, así como los métodos que los reemplazan.

### [Métodos de la clase com.ms.vjsharp.text.FormatDefaults](#)

Documenta el método **com.ms.vjsharp.text.FormatDefaults** compatible con Visual J#.

### [Problemas de compatibilidad con Visual J++ 6.0](#)

Proporciona información detallada sobre problemas de compatibilidad entre Visual J# y Visual J++ 6.0 para varios paquetes de Java y JDK nivel 1.1.4.

## Secciones relacionadas

### [Referencia](#)

Temas de referencia sobre el compilador de Visual J#, el Conversor binario (Jblmp.exe), la compatibilidad de bibliotecas y lenguajes, la actualización de proyectos de Visual J++ 6.0, la sintaxis de Visual J# para la biblioteca de clases de .NET Framework y las características del entorno de desarrollo.

### [Compatibilidad con el lenguaje](#)

Proporciona información de referencia sobre las características de Visual J++ 6.0 y .NET Framework que son compatibles y no compatibles con Visual J#.

# Bibliotecas de clases compatibles

Visual J# está diseñado para proporcionar compatibilidad de bibliotecas de clases funcionalmente equivalente a la mayoría de los paquetes de JDK nivel 1.1.4 incluidos en Visual J++ 6.0. También proporciona las clases especificadas en el currículo de becas avanzadas de informática (Advanced Placement curriculum for Computer Science) del sistema College Board en Estados Unidos. Visual J# admite también bibliotecas WFC (Windows Foundation Classes) y muchos de los siguientes paquetes com.ms.\*:

- **com.ms.lang**
- **com.ms.dll**
- **com.ms.com**
- **com.ms.win32**
- **com.ms.util**
- **com.ms.jdbc.odbc**

**Vea también**

[Compatibilidad con bibliotecas de clases](#) | [Bibliotecas de clases y características no compatibles](#)

# Bibliotecas de clases y características no compatibles

Las siguientes características no son compatibles con Visual J#:

- Tecnologías RMI (Invocación de métodos remotos), JNI (Interfaz nativa de Java) y RNI (Interfaz nativa sin formato).
- Desarrollo de subprogramas. Tampoco es compatible con la ejecución de subprogramas en exploradores.
- Carga de clases desde código de bytes (archivos .class). Sin embargo, permite cargar clases de ensamblados del Lenguaje intermedio de Microsoft (MSIL).
- Variable CLASSPATH. Se proporciona un mecanismo alternativo para buscar y cargar clases y recursos en tiempo de ejecución. Vea [Incompatibilidad con la variable CLASSPATH](#) para obtener más información.

Las siguientes bibliotecas de clases y API no están disponibles en Visual J#:

- Paquetes com.ms.\* que no se enumeran en [Bibliotecas de clases compatibles](#). La documentación de Visual J# proporciona instrucciones para la actualización de muchos de estos paquetes no compatibles. Vea [Actualizar proyectos de Visual J++ 6.0](#) para obtener más información sobre la actualización a bibliotecas de clases de .NET Framework equivalentes.
- Otros paquetes que no formaban parte de las bibliotecas de clases de JDK nivel 1.1.4 que se distribuyeron con Visual J++ 6.0; por ejemplo, sun.\* y netscape.\*.
- No hay compatibilidad con las API de generador de perfiles, control de montones y depuración que se proporcionaban con Visual J++ 6.0. En su lugar, se deben utilizar las API disponibles como parte de las bibliotecas de .NET Framework. Para obtener más información, vea [Depurar y generar perfiles de aplicaciones](#).

## Vea también

[Compatibilidad con bibliotecas de clases](#) | [Características de .NET Framework no compatibles](#) | [Escenarios no compatibles con Visual J#](#) | [Bibliotecas de clases compatibles](#)

# Métodos de la clase `java.lang.Class`

Visual J# admite los siguientes métodos de la clase `java.lang.Class`:

- `public static java.lang.Class Class.FromType(System.Type)`

Convierte una instancia de la clase `System.Type` de .NET Framework en una instancia equivalente de la clase `java.lang.Class`.

- `public static System.Type Class.ToType(java.lang.Class)`

Convierte una instancia de la clase `java.lang.Class` en una instancia equivalente de la clase `System.Type` de .NET Framework

- `public System.Type Class.ToType()`

Devuelve una instancia de la clase `System.Type` de .NET Framework que va a encapsular la instancia de la clase `java.lang.Class` en la que se llama al método.

## Vea también

[Compatibilidad con bibliotecas de clases](#)

# Métodos de la clase **com.ms.vjsharp.text.FormatDefaults**

Visual J# es compatible con el método siguiente de la clase **com.ms.vjsharp.text.FormatDefaults**:

**public static final com.ms.vjsharp.text.FormatDefaults.set(Locale locale, int formatStyle, String datePattern, String timePattern)**

Utilice este método para establecer el modelo de fecha/hora predeterminado para un estilo de formato y una configuración regional. El modelo predeterminado se usará posteriormente al llamar al método **DateFormat.getDateTimeInstance**.

## **Vea también**

[Compatibilidad con bibliotecas de clases](#)



# Métodos de la clase com.ms.wfc.app.Locale

La tabla siguiente muestra los métodos no admitidos de la clase **com.ms.wfc.app.Locale**, así como los nuevos métodos que deberían usarse en su lugar. Debe utilizar los nuevos métodos, tanto al actualizar aplicaciones de Visual J++ a Visual J# como al escribir nuevas aplicaciones que usen esta clase.

Método no admitido	Método de reemplazo
com.ms.wfc.app.Locale.getCountryCode	com.ms.wfc.app.Locale.getCountryRegionCode
com.ms.wfc.app.Locale.getCountry	com.ms.wfc.app.Locale.getCountryRegion
com.ms.wfc.app.Locale.getAbbrevCountry	com.ms.wfc.app.Locale.getAbbrevCountryRegion
com.ms.wfc.app.Locale.getNativeCountry	com.ms.wfc.app.Locale.getNativeCountryRegion
com.ms.wfc.app.Locale.getDefaultCountry	com.ms.wfc.app.Locale.getDefaultCountryRegion
com.ms.wfc.app.Locale.getEnglishCountry	com.ms.wfc.app.Locale.getEnglishCountryRegion
com.ms.wfc.app.Locale.getISO3116CountryName	com.ms.wfc.app.Locale.getISO3116CountryRegionName

Los nuevos métodos introducidos tienen la misma firma que los métodos no admitidos a los que reemplazan, y ofrecen una funcionalidad similar.

## Vea también

[Compatibilidad con bibliotecas de clases](#)

# Problemas de compatibilidad con Visual J+ + 6.0

Esta sección contiene información sobre compatibilidad para Visual J# y Visual J+ + 6.0 relacionada con la compatibilidad con bibliotecas de clases:

- [java.net](#) (Paquete)
- [com.ms.com](#) (Paquete)

## Vea también

[Compatibilidad con bibliotecas de clases](#)

## java.net (Paquete)

Visual J# admite únicamente los protocolos FTP, HTTP y de archivo. No es compatible con los protocolos SMTP, NNTP ni MailTo.

### Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

## com.ms.com (Paquete)

Las siguientes clases del paquete **com.ms.com** no son compatibles:

- **com.ms.com.LicenseMgr**
- **com.ms.com.LICINFO**
- **com.ms.com.StdCOMClassObject**

**Vea también**

[Problemas de compatibilidad con Visual J++ 6.0](#)

# Semántica de seguridad para aplicaciones escritas en Visual J#

En la versión actual, todos los componentes y aplicaciones escritos con Visual J# deben ser totalmente de confianza para que se puedan ejecutar. Si una de estas aplicaciones o componentes se ejecuta en un grupo de código al que la directiva de seguridad no haya concedido el conjunto de permisos denominado FullTrust, se inicia una excepción de seguridad. Esto se aplica a las aplicaciones nuevas escritas con Visual J# donde sólo se utilizan las bibliotecas de clases de .NET Framework, así como a aplicaciones de Visual J++ existentes actualizadas a Visual J#.

Por tanto, considere los siguientes aspectos a la hora de escribir aplicaciones utilizando Visual J#:

- Todas las aplicaciones ejecutadas desde el equipo local en el grupo de código MyComputer obtienen el conjunto de permisos FullTrust por parte de la directiva de seguridad predeterminada de .NET Framework. Por tanto, las aplicaciones ejecutadas desde el equipo local no causan problemas.

Las aplicaciones ejecutadas desde grupos de código de ubicaciones remotas (como un recurso compartido de red) en Internet o una intranet. Puesto que las aplicaciones ejecutadas en estos grupos de código no obtienen el conjunto de permisos FullTrust por parte de la directiva de seguridad predeterminada de .NET Framework, no funcionan e inician una excepción de seguridad.

Una posible solución consiste en hacer que las aplicaciones o los componentes se ejecuten desde ubicaciones remotas totalmente de confianza. Para ser considerado totalmente de confianza por la directiva de seguridad de .NET Framework, se debe dar alguno de los siguientes casos:

- El componente o aplicación se ha firmado con un par de claves y se ha modificado la directiva de seguridad de .NET Framework en el equipo para conceder el conjunto de permisos denominado FullTrust a todos los componentes o aplicaciones firmados con este par de claves.
- El componente o aplicación se ha firmado con un certificado Authenticode y se ha modificado la directiva de seguridad de .NET Framework en el equipo para conceder el conjunto de permisos denominado FullTrust a todos los componentes o aplicaciones firmados con este certificado.
- Se ha modificado la directiva de seguridad de .NET Framework en el equipo para conceder el conjunto de permisos denominado FullTrust a controles descargados desde el sitio Web (URL) donde se aloja el componente o aplicación.
- Todos los controles administrados alojados en Internet Explorer se ejecutan en grupos de código de Internet o una intranet y, por tanto, no funcionan e inician una excepción de seguridad. Esto mismo ocurre cuando se descarga y ejecuta una aplicación de Visual J# utilizando la característica de descarga de código de Internet Explorer.
- Si se modifica la directiva de seguridad del grupo de código MyComputer para cambiar del nivel predeterminado de FullTrust a un conjunto de permisos más restringido, las aplicaciones iniciadas desde el equipo local dejan de ejecutarse.

# Referencia para la actualización a Visual J#

En esta sección se proporcionan temas de ayuda para problemas detectados por el Asistente para actualización durante la actualización de proyectos de Visual J++ 6.0 a Visual J#.

Los temas de esta sección están relacionados con los problemas identificados durante la actualización de un proyecto de Visual J++ 6.0 a Visual J#. Estos temas están pensados para tener acceso a ellos desde los vínculos del informe de actualización, que se genera durante el proceso de actualización.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#) | [Asistente para actualización a Visual J#](#) | [Informe de actualización de Visual J#](#) | [Conversor binario de Visual J#](#)

# No se admiten proyectos de subprograma

Visual J# no admite la creación de subprogramas. El Asistente para actualización no proporciona tampoco una actualización directa para los proyectos de subprograma de Visual J++ 6.0.

## Vea también

[Bibliotecas de clases y características no compatibles](#)

# La interfaz `java.io.Externalizable` tiene un problema de compatibilidad

Los objetos serializados en Visual J++ 6.0 se pueden deserializar en Visual J# sólo si los campos de los objetos son tipos primitivos o **`java.lang.String`**. Un objeto que haga referencia a cualquiera de las demás clases de los paquetes de JDK nivel 1.1.4 no se puede deserializar. Por supuesto, cualquier objeto serializado con Visual J# se puede deserializar en Visual J#, independientemente de la limitación anterior.

## Vea también

[La interfaz `java.io.Serializable` tiene un problema de compatibilidad](#)



# La interfaz `java.io.Serializable` tiene un problema de compatibilidad

Los objetos serializados en Visual J++ 6.0 se pueden deserializar en Visual J# sólo si los campos de los objetos son tipos primitivos o **`java.lang.String`**. Un objeto que haga referencia a cualquiera de las demás clases de los paquetes de JDK nivel 1.1.4 no se puede deserializar. Por supuesto, cualquier objeto serializado con Visual J# se puede deserializar en Visual J#, independientemente de la limitación anterior.

## Vea también

[La interfaz `java.io.Externalizable` tiene un problema de compatibilidad](#)

# Los métodos de `java.lang.Class` tienen problemas de compatibilidad

Los siguientes métodos de `java.lang.Class` tienen problemas de compatibilidad entre Visual J++ 6.0 y Visual J#:

## **`java.lang.Class.forName(String nombreClase)`**

Este método no utiliza la variable de entorno `CLASSPATH` para buscar clases. Sigue la semántica de Common Language Runtime para buscar y cargar los ensamblados que contienen los tipos correspondientes. Para obtener más información, vea [Actualizar aplicaciones que cargan clases dinámicamente](#).

## **`java.lang.Class.getDeclaredMethods`**

El orden de los elementos devueltos por este método no es el mismo que en Visual J++ 6.0. Es posible que debe modificar las aplicaciones que dependen del orden de los elementos devueltos por este método.

## **`java.lang.Class.getMethods`**

El orden de los elementos devueltos por este método no es el mismo que en Visual J++ 6.0. Es posible que debe modificar las aplicaciones que dependen del orden de los elementos devueltos por este método.

## **`java.lang.Class.getFields`**

El orden de los elementos devueltos por este método no es el mismo que en Visual J++ 6.0. Es posible que debe modificar las aplicaciones que dependen del orden de los elementos devueltos por este método.

## **Vea también**

[Actualizar aplicaciones que utilizan cargadores de clases personalizados](#) |

[Actualizar aplicaciones que cargan clases dinámicamente](#) | [Problemas de compatibilidad con Visual J++ 6.0](#)

# Problema de compatibilidad de java.lang.ClassLoader

No hay compatibilidad con **ClassLoader** para convertir códigos de bytes en un objeto **Class**. En Visual J#, ya no son compatibles los siguientes métodos:

- **ClassLoader.defineClass**
- **ClassLoader.resolveClass**

Estos métodos inician **com.ms.vjsharp.MethodNotSupportedException** en Visual J#.

Para obtener más información, vea [Actualizar aplicaciones que utilizan cargadores de clases personalizados](#).

# La clase `java.lang.Compiler` no se admite

Visual J# no admite la clase `java.lang.Compiler`.

## Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

# La clase `java.text.CollationElementIterator` no se admite

Visual J# no admite la clase `java.text.CollationElementIterator`.

# Algunos métodos de `java.text.RuleBasedCollator` no se admiten

No se admite el constructor **`RuleBasedCollator(String)`** de esta clase.

Tampoco se admiten los siguientes métodos de esta clase:

- **`RuleBasedCollator.getRules`**
- **`RuleBasedCollator.getCollationElementIterator`**

# Algunos métodos de `java.lang.Runtime` tienen problemas de compatibilidad

Los métodos siguientes tienen problemas de compatibilidad con Visual J++ 6.0:

- **`java.lang.Runtime.freeMemory`**

Visual J# devuelve el número de bytes de memoria libre disponible, mientras que Visual J++ 6.0 devuelve el número de kilobytes. El comportamiento de Visual J# cumple las especificaciones de las bibliotecas de clases de Java.

- **`java.lang.Runtime.totalMemory`**

Visual J# devuelve el número de bytes de memoria total disponible, mientras que Visual J++ 6.0 devuelve el número de kilobytes. El comportamiento de Visual J# cumple las especificaciones de las bibliotecas de clases de Java.

- **`java.lang.Runtime.load`**

Los siguientes métodos no son compatibles con Visual J#:

- **`java.lang.Runtime.traceInstructions`**

- **`java.lang.Runtime.traceMethodCalls`**

Estos métodos inician **`com.ms.vjsharp.MethodNotSupportedException`** en Visual J#.

## Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

# La clase `java.lang.SecurityManager` tiene problemas de compatibilidad

Los métodos de esta clase que tratan con **CallStack** no son compatibles con Visual J#. Estos métodos son:

- **classDepth**
- **classLoaderDepth**
- **currentClassLoader**
- **currentLoadedClass**
- **getClassContext**
- **inClass**
- **inClassLoader**

Estos métodos inician **com.ms.vjsharp.MethodNotSupportedException** en Visual J#.

Vea [Actualizar aplicaciones que utilizan la semántica de seguridad de Visual J++ 6.0](#) para obtener más información.



# El método `java.lang.Thread.getName` tiene un problema de compatibilidad

El método **`java.lang.Thread.getName`** tiene un problema de compatibilidad en Visual J#. Vea [Problemas de compatibilidad con Visual J++ 6.0](#).

# El método `java.lang.Thread.dumpStack` no se admite

Este método inicia **`com.ms.vjsharp.MethodNotSupportedException`** en Visual J#.

## Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

# El método `java.lang.Thread.countStackFrames` no se admite

Este método inicia `com.ms.vjsharp.MethodNotSupportedException` en Visual J#.

## Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

# El método `java.lang.ThreadGroup.allowThreadSuspension` no se admite

Este método inicia **`com.ms.vjsharp.MethodNotSupportedException`** en Visual J#.

## Vea también

[Problemas de compatibilidad con Visual J++ 6.0](#)

# El paquete `java.security` tiene problemas de compatibilidad

El proveedor de seguridad predeterminado proporcionado con Visual J# tiene los siguientes problemas de compatibilidad con el proveedor predeterminado de Visual J++ 6.0:

No se admite la codificación de claves en las claves generadas por el proveedor de seguridad predeterminado. Por tanto, los siguientes métodos de `java.security` no son compatibles con el proveedor predeterminado de Visual J#:

- `java.security.Key.getEncoded`
- `java.security.Key.getEncoded`

Estos métodos inician `com.ms.vjsharp.MethodNotSupportedException` en Visual J#.

- El proveedor de seguridad predeterminado no admite claves generadas por otros proveedores de seguridad. Esto significa que no es posible firmar ni comprobar datos utilizando claves públicas y privadas de otros fabricantes.
- El proveedor de seguridad predeterminado no puede comprobar firmas generadas por otros proveedores de seguridad.

Sin embargo, Visual J# permite a los usuarios conectar proveedores de seguridad de otros fabricantes para operaciones como la generación de claves y firmas.

Para obtener más información, vea [Actualizar aplicaciones que utilizan proveedores de seguridad de otros fabricantes](#).

# El paquete com.ms.ActiveX no se admite

Visual J# no permite alojar controles ActiveX. Para obtener más información, vea [Alojar controles ActiveX en Java](#)

# El paquete com.ms.asp no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# El paquete com.ms.awt no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).



# La clase `com.ms.com.ComSuccessException` tiene un problema de compatibilidad

En proyectos actualizados de Visual J+ + 6.0, los HRESULT que no fallan como S\_FALSE se tratan siempre como equivalentes de S\_OK y no se inician excepciones de tipo **`com.ms.com.ComSuccessException`**.

**Vea también**

[Java llama a COM](#)

# La interfaz com.ms.com.NoAutoMarshaling tiene un problema de compatibilidad

En Visual J#, no se admiten las interfaces de marcador **com.ms.com.NoAutoMarshaling** y **com.ms.com.NoAutoScripting** de contenedores JActiveX. Puede ser necesario modificar durante la actualización las aplicaciones de Visual J++ que dependen de la funcionalidad proporcionada por estas interfaces.

Para obtener más información, vea [Problemas y soluciones: Java llama a COM](#).

## Vea también

[Java llama a COM](#)

# El paquete com.ms.debug no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.directX no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.ie no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.iis no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# El paquete `com.ms.io.clientstorage` no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.io no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).



# El paquete com.ms.license no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.mtx no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# El paquete com.ms.net.wininet no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.object no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete `com.ms.object.dragdrop` no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.security no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# El paquete `com.ms.security.permissions` no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# El paquete com.ms.service no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).



# El paquete `com.ms.util.InputMethod` no se admite

Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.vm no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# La clase `com.ms.wfc.app.Locale` tiene problemas de compatibilidad

Ya no se admiten los siguientes métodos de la clase `com.ms.wfc.app.Locale`:

- `com.ms.wfc.app.Locale.getCountryCode`
- `com.ms.wfc.app.Locale.getCountry`
- `com.ms.wfc.app.Locale.getAbbrevCountry`
- `com.ms.wfc.app.Locale.getNativeCountry`
- `com.ms.wfc.app.Locale.getDefaultCountry`
- `com.ms.wfc.app.Locale.getEnglishCountry`
- `com.ms.wfc.app.Locale.getISO3116CountryName`

Al actualizar aplicaciones de Visual J++ a Visual J# los programadores deben usar los nuevos métodos que los reemplazan. Para obtener más información vea [Métodos de la clase in `com.ms.wfc.app.Locale`](#).

## Vea también

[Métodos de la clase `com.ms.wfc.app.Locale`](#)

# No se admiten proyectos de controles ActiveX

En Visual J#, no se admite la exposición de controles WFC o Java Beans como controles ActiveX. Si un proyecto de Visual J++ es un control ActiveX, no funcionará después de actualizarlo a Visual J#.

## Vea también

[COM llama a Java](#)

# El paquete com.ms.xml no se admite

Los paquetes com.ms.xml.\* siguientes no son compatibles con Visual J#:

- **com.ms.xml.dso**
- **com.ms.xml.om**
- **com.ms.xml.parser**
- **com.ms.xml.util**

Deberá modificar la aplicación para reemplazar las clases de estos paquetes con clases que tengan funcionalidad equivalente en el espacio de nombres **System.Xml** de las bibliotecas de .NET Framework.

Para obtener más información, vea [Actualizar componentes que utilizan el paquete com.ms.xml](#).

# Problema de compatibilidad del método com.ms.com.ComLib.ownsCleanup

Este método devuelve siempre true en Visual J#.

# Problema de compatibilidad del método `com.ms.dll.DllLib.copy`

El método `com.ms.dll.DllLib.copy` tiene un problema de compatibilidad con Visual J#. Vea [Problemas de compatibilidad con Visual J++ 6.0](#).

# Los métodos IME (Editor de métodos de entrada) de `com.ms.lang.SystemX` no son compatibles

Los métodos siguientes de la clase `com.ms.lang.SystemX` no son compatibles:

- `getDefaultInputManager`
- `getInputManager`
- `setInputManager`
- `setKeyBoardLanguage`

Estos métodos inician `com.ms.vjsharp.MethodNotSupportedException` en Visual J#.



# Problema de compatibilidad de la clase com.ms.com.Variant

Visual J# no admite el cálculo de referencias de VARIANTS de tipo VT\_BYREF.

**Vea también**

[Java llama a COM](#)

# Problema de compatibilidad de la interfaz **com.ms.com.NoAutoScripting**

En Visual J#, no se admiten las interfaces de marcador **com.ms.com.NoAutoMarshaling** y **com.ms.com.NoAutoScripting** de contenedores JActiveX. Puede ser necesario modificar las aplicaciones de Visual J++ que dependen de la funcionalidad proporcionada por estas interfaces.

Para obtener más información, vea [Problemas y soluciones: Java llama a COM](#).

## Vea también

[Java llama a COM](#)

# El paquete com.ms.security.auditing no se admite

Para obtener más información, vea [Actualizar paquetes com.ms.\\* a bibliotecas de clases de .NET Framework equivalentes](#).

# La clase **com.ms.util.SystemVersionManager** no se admite

Visual J# no admite la clase **com.ms.util.SystemVersionManager**.

# La clase **com.ms.util.IncludeExcludeWildcards** no se admite

Visual J# no admite la clase **com.ms.util.IncludeExcludeWildcards**.

# La clase **com.ms.util.SetComparer** no se admite

Visual J# no admite la clase **com.ms.util.SetComparer**.

# Problema de compatibilidad de la clase `com.ms.util.IntRanges`

En Visual J# no se admiten los siguientes métodos de la clase `com.ms.util.IntRanges`:

- `IntRanges.condense(IIntRangeComparator judge)`
- `IntRanges.intersect(IntRanges other, IIntRangeComparator judge)`
- `IntRanges.removeRange(int i, IIntRangeComparator hook)`
- `IntRanges.removeRange(int s, int e, IIntRangeComparator hook)`
- `IntRanges.removeRanges(int s, int count, IIntRangeComparator hook)`
- `IntRanges.removeSingleton(int n, IIntRangeComparator hook)`
- `IntRanges.sort(IIntRangeComparator judge)`
- `IntRanges.compare (IntRanges intRange1, IIntRangeComparator judge)`

# La clase **com.ms.util.UnsignedIntRanges** no se admite

Visual J# no admite la clase **com.ms.util.UnsignedIntRanges**.



# La clase **com.ms.util.WildcardExpression** no se admite

Visual J# no admite la clase **com.ms.util.WildcardExpression**.

# Problemas de compatibilidad de la clase **com.ms.util.FileVersionInformation**

La clase **com.ms.util.FileVersionInformation** tiene un problema de compatibilidad en Visual J#. Esta clase no admite las propiedades **FileType** y **OperatingSystem** en Visual J#. Vea [Problemas de compatibilidad con Visual J++ 6.0](#).

# La interfaz `com.ms.util.IWildcardExpressionComparator` no se admite

Visual J# no admite la interfaz `com.ms.util.IWildcardExpressionComparator`.

# La interfaz **com.ms.util.IIntRangeComparator** no se admite

Visual J# no admite la interfaz **com.ms.util.IIntRangeComparator**.

# Problema de compatibilidad de la clase `com.ms.util.Timer`

En Visual J# no se admiten los siguientes constructores de la clase `com.ms.util.Timer`:

- `public Timer(TaskManager tm, long p)`
- `public Timer(TaskManager tm, long p, int id)`
- `public Timer(TaskManager tm, long p, boolean r)`
- `public Timer(TaskManager tm, long p, int id, boolean r)`
- `public Timer(TaskManager tm, TimeListener defaultListener, long p)`
- `public Timer(TaskManager tm, TimeListener defaultListener, long p, int id)`
- `public Timer(TaskManager tm, TimeListener defaultListener, long p, boolean r)`
- `public Timer(TaskManager tm, TimeListener defaultListener, long p, int id, boolean r)`

# La interfaz **com.ms.util.ProvideSetComparisonInfo** no se admite

Visual J# no admite la interfaz **com.ms.util.ProvideSetComparisonInfo**.

# Los métodos `com.ms.util. HTMLTokenizer.mark` y `com.ms.util. HTMLTokenizer.reset` no se admiten

Estos métodos inician `com.ms.vjsharp.MethodNotSupportedException` en Visual J#.

# La clase **com.ms.util.Task** no se admite

Visual J# no admite la clase **com.ms.util.Task**. Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).



# La clase **com.ms.util.TaskManager** no se admite

Visual J# no admite la clase **com.ms.util.TaskManager**. Para obtener más información, vea [Bibliotecas de clases y características no compatibles](#).

# El paquete com.ms.wfc.html tiene problemas de compatibilidad

Los proyectos creados como proyectos HTML de código subyacente en Visual J+ + 6.0 tienen un problema de compatibilidad. Para obtener más información, vea [Actualizar componentes que utilizan el paquete com.ms.wfc.html](#).

# La búsqueda de recursos tiene un problema de compatibilidad

El Asistente para actualización determinó que el proyecto de Visual J++ 6.0 utiliza recursos. En Visual J#, hay algunos cambios en el modo en que se buscan los recursos en tiempo de ejecución. Esto afecta tanto a recursos WFC (utilizados por proyectos del Asistente para aplicaciones) como a los recursos especificados por el usuario que se busquen con el método **java.lang.Class.getResource**.

Para que la aplicación busque los recursos correctamente en Visual J#, éstos se deben convertir a un nuevo formato utilizado por los archivos de recursos de aplicaciones orientadas a .NET Framework.

El asistente convierte automáticamente los archivos de recursos del proyecto con las extensiones '.resources' y '.properties' al nuevo formato y los guarda con la extensión '.resx'. Para estos recursos, no tiene que hacer nada.

Para los archivos de recursos del proyecto de Visual J++ 6.0 que tengan cualquier otra extensión y los recursos cargados desde una ubicación fuera de la carpeta del proyecto, especificada con CLASSPATH, debe convertir los recursos al formato utilizado por los archivos de recursos de Visual Studio .NET.

Para obtener más información, vea [Utilizar recursos en Visual J#](#).

# Paquetes de otros fabricantes no compatibles con esta versión

Los siguientes paquetes estaban disponibles con Visual J++ 6.0, pero no lo están en Visual J#.

- **netscape.javascript**
- **sun.audio**
- **sun.awt.image**
- **sun.beans.editors**
- **sun.beans.infos**
- **sun.io**
- **sun.misc**
- **sun.net.ftp**
- **sun.net**
- **sun.net.nntp**
- **sun.net.smtp**
- **sun.net.www**
- **sun.net.www.protocol.doc**
- **sun.net.www.protocol.file**
- **sun.net.www.protocol.ftp**
- **sun.net.www.protocol.http**
- **sun.security.acl**
- **sun.security.pkcs**
- **sun.security.util**
- **sun.security.x509**
- **sun.tools.jar**

Las aplicaciones que utilizan clases de estos paquetes se deberán modificar para que utilicen la funcionalidad equivalente de las bibliotecas de clases de .NET Framework, o bibliotecas de otros fabricantes orientadas a Common Language Runtime.

# Problema de compatibilidad de la directiva **@com.register**

Existen incompatibilidades con Visual J++ 6.0 en el conjunto de interfaces expuestas por contenedores COM a los que se puede llamar (CCW) de componentes de Java. En Visual J#, los CCW no exponen las siguientes interfaces:

- **IProvideClassInfo2** e **IExternalConnection**
- **IPersist**, **IPersistStreamInit** e **IPersistStorage** (expuestas en Microsoft Java Virtual Machine cuando una clase implementa **java.io.Serializable** o **java.io.Externalizable**)
- Las interfaces de marcador **com.ms.com.NoAutoMarshaling** y **com.ms.com.NoAutoScripting** de la clase se omiten. Esto puede dar lugar a problemas de compatibilidad en tiempo de ejecución con respecto a Visual J++ 6.0.
- Actualmente, no existe compatibilidad para recibir eventos desencadenados por clases de Java o COM en clientes COM.

## Vea también

[Soluciones: COM llama a Java](#) | [Java llama a COM](#) | [Compatibilidad parcial con directivas @com](#)

# La directiva **@com.transaction** no se admite

La directiva **@com.transaction** se omite en Visual J#.

## Vea también

[COM llama a Java](#) | [Compatibilidad parcial con directivas @com](#)

# La directiva **@com.typeinfo** no se admite

La directiva **@com.typeinfo** se omite en Visual J#.

## Vea también

[COM llama a Java](#) | [Compatibilidad parcial con directivas @com](#)

# El cálculo de referencias personalizado no se admite en la interoperabilidad de Java y COM ni en J/Direct

Microsoft Visual J# no admite contenedores JActiveX que utilicen cálculo personalizado de referencias para convertir tipos Java en tipos nativos. La compilación de estos contenedores con el compilador de Visual J# da lugar a errores en tiempo de compilación.

Para obtener más información, vea [Actualizar componentes de Java y COM que utilizan cálculo de referencias personalizado](#).

## Vea también

[Compatibilidad parcial con directivas @com](#)



# Posible problema de modelo de subprocesamiento al alojar controles ActiveX en formularios WFC

Cuando se alojan controles ActiveX en formularios WFC, puede ser necesario definir el modelo de apartamento del subproceso que aloja el control ActiveX como un apartamento de un único subproceso (STA). Este requisito se logra en gran parte adjuntando el atributo `System.STAThread` al método estático público `void main(String[] args)` de la clase.

Para obtener más información, vea [Controlar el modelo de subprocesamiento de los componentes COM](#).

# La tecnología RNI no se admite

La tecnología RNI (Interfaz nativa sin formato) no es compatible con Visual J#.

## Vea también

[Bibliotecas de clases y características no compatibles](#)

# La tecnología JNI no se admite

La tecnología JNI (Interfaz nativa de Java) no es compatible con Visual J#.

## Vea también

[Bibliotecas de clases y características no compatibles](#)

# La palabra clave volatile no se admite

La palabra clave volatile no es compatibles con Visual J#:

## Vea también

[Bibliotecas de clases y características no compatibles](#)

# No se encontró la biblioteca de tipos para la clase o interfaz

Durante el análisis de actualización de los archivos del proyecto, el Asistente para actualización no encontró la biblioteca de tipos de una interfaz o una clase COM utilizada en el proyecto. El asistente localiza esta biblioteca de tipos buscando el GUID de la interfaz o clase COM en el Registro del equipo. Vea la información detallada que ofrece el asistente sobre este problema para obtener más información sobre los GUID para los que no se encontró información de registro.

La búsqueda de GUID puede fallar porque hay una interfaz no registrada. Es posible que esto no sea un problema después de la actualización, si el proyecto actualizado hace referencia a la clase que implementa la interfaz.

Sin embargo, si la búsqueda de GUID falla porque no se encuentra el CLSID de una clase, dará lugar a un error durante la generación del proyecto actualizado. Para corregir esto, asegúrese de que el proyecto de Visual J++ 6.0 se puede generar y ejecutar correctamente en el equipo donde va a llevar a cabo la actualización.

## Vea también

[Actualizar desde Visual J++ 6.0](#)

# Referencias de clase o interfaz sin resolver

El Asistente para actualización determinó que el proyecto de Visual J++ 6.0 hace referencia a clases o interfaces que no están definidas en el proyecto, ni en las referencias de ensamblado proporcionadas en la página [Agregar referencias del proyecto](#) del Asistente para actualización. Vea la información detallada del informe sobre este problema para obtener más información sobre las clases o tipos cuyas referencias no se encontraron.

Además de buscar entre las referencias del proyecto de Visual J++ 6.0 y de ensamblado especificadas durante la actualización, el análisis de actualización busca la referencia que falta en el directorio y las ubicaciones de archivos definidas en el valor classpath del proyecto de Visual J++ 6.0.

Este problema puede surgir cuando el proyecto de Visual J++ 6.0 depende de clases definidas en otro proyecto de la solución o en bibliotecas de clases externas a las que se hace referencia utilizando la variable de entorno CLASSPATH del sistema.

Para resolverlo, puede agregar los ensamblados que contienen las clases o interfaces sin resolver a la lista de referencias del proyecto actualizado en Visual J#. Para obtener más información sobre el modo de hacer esto, vea [Agregar y quitar referencias de proyecto](#).

Como alternativa, durante la actualización, puede proporcionar una referencia a los ensamblados que contienen las clases o interfaces sin resolver. Para obtener más información sobre el modo de hacer esto, vea [Asistente para actualizar: agregar referencias del proyecto](#).

Si no tiene los archivos de código fuente de las clases o interfaces a las que se hace referencia externamente, pero tiene los archivos binarios y de almacenamiento, puede convertir los archivos de clase (.class) o de almacenamiento (.cab, .jar o .zip) en ensamblados MSIL utilizando la **herramienta** [Conversor binario de Visual J#](#) (jbimp.exe).

**Nota** Tenga en cuenta que, puesto que no se encontraron algunas referencias, el análisis de actualización puede ser incompleto para clases o interfaces que extienden o implementan las clases o interfaces que faltan.

# Referencias de clase o interfaz sin resolver en classpath

El Asistente para actualización determinó que el proyecto de Visual J++ 6.0 tiene referencias a clases o interfaces ubicadas en la variable classpath definida en el proyecto. Esta determinación se lleva a cabo buscando el archivo de clase (.class) con un nombre que coincide con las clases o interfaces sin resolver.

Vea la información detallada sobre este problema para identificar las ubicaciones classpath (directorios o archivos de almacenamiento) donde se han ubicado las referencias que faltan.

Para resolver las referencias que faltan, debe crear uno o más ensamblados que contengan definiciones de las clases o interfaces que faltan y agregar sus referencias al proyecto actualizado.

## Si tiene archivos de código fuente

Si tiene los archivos de código fuente de las referencias de las clases o interfaces que faltan, puede generar una biblioteca de clases a partir de los archivos de código fuente utilizando un proyecto de biblioteca de clases de Visual J#. Este archivo de biblioteca de clases (.dll) se puede agregar a las referencias del proyecto actualizado. Utilice las ubicaciones de classpath identificadas en el informe de actualización para identificar las clases que deben formar parte de esta biblioteca.

## Si no tiene archivos de código fuente

Si no tiene los archivos de código fuente, pero tiene los archivos binarios o de almacenamiento, puede convertir los archivos de clase (.class) o de almacenamiento (.cab, .jar o .zip) en un ensamblado MSIL utilizando la **herramienta Conversor binario de Visual J#** (jbimp.exe). Este ensamblado se puede agregar a la lista de referencias del proyecto actualizado. Utilice las ubicaciones de classpath identificadas en el informe de actualización para identificar las clases que deben formar parte de este ensamblado.

## Vea también

[Agregar y quitar referencias de proyecto](#)

# No se pudo agregar al proyecto la referencia a la biblioteca de tipos

Durante el análisis de actualización, el asistente detectó que el proyecto hace referencia a una biblioteca de tipos COM. Sin embargo, no se pudo agregar una referencia a esta biblioteca (.dll o .tlb). El informe de actualización indica las bibliotecas de tipos que no se pudieron agregar.

Para corregir esto, agregue manualmente una referencia a esta biblioteca de tipos utilizando Visual J#. Para obtener más información, vea [Agregar y quitar referencias de proyecto](#).



# Configuración del control de código fuente no actualizada al nuevo proyecto

El Asistente para actualización determinó que el proyecto de Visual J++ 6.0 estaba bajo control de código fuente. Este asistente no actualiza la configuración de código fuente del archivo de proyecto de Visual J++ 6.0 al archivo de proyecto de Visual J# actualizado.

Utilice el comando Cambiar control de código fuente del menú Control de código fuente para volver a enlazar los proyectos con sus ubicaciones de servidor. Para obtener más información, vea [Cambiar conexiones](#).

## Vea también

[Agregar soluciones y proyectos al control de código fuente](#)

# Visual J# no admite Classpath

El proyecto de Visual J++ 6.0 tiene un valor **Classpath** específico del proyecto. Esta propiedad se utilizaba para buscar clases a las que hace referencia el proyecto durante el análisis de actualización. Sin embargo, Visual J# no admite **Classpath** y no se actualiza este valor.

## Vea también

[Incompatibilidad con la variable CLASSPATH](#)

# Las configuraciones anteriores y posteriores a la generación no se actualizan

Las configuraciones anteriores y posteriores a la generación del proyecto de Visual J++ 6.0 no se actualizan al nuevo proyecto. Los proyectos de Visual J# no admiten configuraciones anteriores y posteriores a la generación.

Para reemplazar estas configuraciones, puede crear pasos de generación personalizada para el proyecto actualizado. Si desea crear un paso de generación personalizada en un proyecto de Visual J#, agregue un proyecto de archivo MAKE de C++ a la solución (vea [Crear un proyecto de archivos MAKE](#)) y cambie la propiedad **Tipo de configuración** a Utilidad (vea [Página de propiedades General](#) para obtener más información). Vea [Introducción a los pasos de generación personalizada y los eventos de generación](#) para obtener más información.

# La configuración de bibliotecas de tipos del proyecto no se actualiza

La configuración para la creación de bibliotecas de tipos del proyecto de Visual J+ + 6.0 no se actualiza a Visual J#. Los proyectos de Visual J# no admiten la creación de bibliotecas de tipos como parte del resultado del proyecto.

# El proyecto se ejecuta utilizando un programa de inicio personalizado

El proyecto de Visual J++ 6.0 tiene una configuración de programa de inicio personalizado. Se utiliza un programa externo especificado en esta configuración para iniciar la aplicación cuando se ejecuta desde el entorno de desarrollo de Visual J++ 6.0.

En Visual J#, es posible que no funcione el uso del mismo programa externo para iniciar la aplicación, puesto que los resultados del proyecto son ahora ensamblados administrados y no archivos de clase.

Deberá definir manualmente la propiedad Aplicación de inicio del proyecto actualizado para establecer un programa externo correcto para iniciar el proyecto desde el entorno de desarrollo de Visual J#. Para obtener más información, vea [Depurar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#).

## Error al actualizar el archivo de recursos

El proyecto de Visual J++ 6.0 contiene archivos de recursos con el formato de Visual J++ 6.0 (con las extensiones de archivo .resources y .properties). El Asistente para actualización intentó convertirlos al formato de archivo de recursos .NET (.resx), pero la conversión falló de modo inesperado.

Para resolver este problema, puede convertir manualmente el archivo de recursos y agregarlo al proyecto actualizado. Para obtener más información, vea [Utilizar recursos en aplicaciones de Visual J#](#).

# El análisis de los archivos de proyecto no se pudo completar

El Asistente para actualización analiza el proyecto de Visual J++ 6.0 para detectar problemas de actualización. Durante el análisis, se encontró un error que dio lugar a que no se completara el análisis de actualización. Por tanto, es posible que no se hayan detectado y agregado al informe de actualización todos los problemas de actualización. Puede que la generación y ejecución del proyecto actualizado no sean correctas a menos que se detecten y corrijan todos los problemas de actualización.

La causa del error durante el análisis puede ser alguna de las siguientes:

## Errores de generación en el proyecto de Visual J++ 6.0

Si el proyecto de Visual J++ 6.0 contiene errores de sintaxis u otros errores que darán lugar a que falle la generación en Visual J++ 6.0, también dará lugar a que falle el análisis de actualización. Para lograr una actualización correcta, asegúrese de que el proyecto se genera y ejecuta en Visual J++ 6.0 y, a continuación, intente actualizarlo de nuevo.

## Faltan referencias durante la actualización

Si el proyecto hace referencia a clases u objetos COM externos al proyecto, hay una posibilidad de que estas referencias que faltan dieran lugar a que no se completara el análisis de actualización. Se recomienda que actualice de nuevo el proyecto de Visual J++ 6.0, agregando todas las referencias externas en la página "Agregar referencias" del Asistente para actualización. Para obtener más información sobre el modo de hacer esto, vea [Asistente para actualización: agregar referencias del proyecto](#).

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#)

# Problemas al actualizar el archivo de proyecto al formato nuevo

El Asistente para actualización encontró problemas al actualizar el archivo de proyecto de Visual J++ 6.0 a Visual J#. Esto puede deberse a diferencias en las propiedades del proyecto que se almacenan en un archivo de proyecto de Visual J++ 6.0 y de Visual J#. Vea los detalles del problema en el informe de actualización dentro de esta categoría para obtener información más específica.



# Problemas de actualización en las bibliotecas de clases de JDK nivel 1.1.4

El Asistente para actualización detectó problemas en los archivos de código fuente del proyecto que implican el uso de bibliotecas de clases de JDK nivel 1.1.4. Algunos de los métodos y clases compatibles con bibliotecas de clases de Visual J# pueden tener problemas de compatibilidad con Visual J++ 6.0. Vea los detalles del problema en el informe de actualización dentro de esta categoría para obtener información más específica.

Para obtener más información sobre la compatibilidad de bibliotecas de clases en Visual J#, vea [Compatibilidad con bibliotecas de clases](#).

## Vea también

[Bibliotecas de clases y características no compatibles](#) | [Problemas de compatibilidad con Visual J++ 6.0](#)

# Problemas de actualización en las extensiones de Microsoft

El Asistente para actualización detectó problemas en el uso de extensiones de Microsoft en Visual J++ 6.0, como clases WFC y otros paquetes de la jerarquía com.ms.\*. Estos problemas se deben a que algunas de estas extensiones tienen problemas de compatibilidad en Visual J#. Vea los detalles del problema en el informe de actualización dentro de esta categoría para obtener más información.

## Vea también

[Compatibilidad con bibliotecas de clases](#)

## Faltan extensiones de otros fabricantes en Visual J#

El Asistente para actualización detectó el uso de extensiones de otros fabricantes en paquetes de JDK nivel 1.1.4. Estas extensiones (como paquetes netscape.\* y sun.\*) estaban disponibles en Visual J++ 6.0, pero no se admiten en Visual J#. Para obtener más información, vea los detalles del problema en el informe de actualización dentro de esta categoría.

### **Vea también**

[Bibliotecas de clases y características no compatibles](#)

# Se encontraron referencias COM sin resolver en la actualización

El Asistente para actualización detectó referencias a componentes COM en el proyecto que no se pudieron resolver. Para obtener más información, vea los detalles del problema en el informe de actualización dentro de esta categoría.

Una de las causas puede ser que el componente COM al que se hace referencia no está disponible en el sistema donde se está realizando la actualización. Para corregir esto, asegúrese de que el proyecto de Visual J++ 6.0 se puede generar y ejecutar en el mismo sistema donde lo va a actualizar a Visual J#.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#)

# Se encontraron referencias sin resolver en la actualización

El Asistente para actualización detectó que algunas referencias a clases o interfaces de los archivos de código fuente del proyecto no estaban resueltas. Para obtener más información, vea los detalles del problema en el informe de actualización dentro de esta categoría.

## Vea también

[Actualizar proyectos de Visual J++ 6.0](#)

# Otros problemas varios

Problemas de actualización generales detectados por el Asistente para actualización. Para obtener más información, vea los detalles del problema en el [informe de actualización de Visual J#](#) para esta categoría.

## Vea también

[Actualizar desde Visual J++ 6.0](#)

# Herramientas de Visual J#

Visual J# incluye herramientas para facilitar la creación e implementación de aplicaciones orientadas a .NET Framework. Esta sección contiene información acerca de estas herramientas.

## En esta sección

### [Conversor binario de Visual J# \(para código de bytes de Java en conversiones MSIL\)](#)

Contiene información sobre el conversor binario de Visual J# (Jblmp.exe), una herramienta que convierte la mayoría del código de bytes de Java de JDK nivel 1.1.4 en Lenguaje intermedio de Microsoft (MSIL).

## Secciones relacionadas

### [Referencia](#)

Contiene temas sobre el compilador de Visual J#, la compatibilidad de bibliotecas y lenguajes, la actualización de proyectos de Visual J++ 6.0, la sintaxis de Visual J# para la biblioteca de clases de .NET Framework y las características del entorno de desarrollo.

# Conversor binario de Visual J# (para código de bytes de Java en conversiones MSIL)

El conversor binario de Visual J# (JbImp.exe) convierte determinados archivos de código de bytes de Java (.class) en Lenguaje intermedio de Microsoft® (MSIL). Esta herramienta permite a los desarrolladores convertir la mayoría de las bibliotecas y aplicaciones de JDK nivel 1.1.4 disponibles sólo como archivos de código de bytes en ensamblados de MSIL y ejecutarlos en Visual J#. Sólo se puede utilizar esta herramienta si no están disponibles los archivos de código fuente de Java para las aplicaciones o bibliotecas. Si están disponibles los archivos de código fuente de Java, se recomienda utilizar el compilador de Visual J# (vjc.exe) en su lugar.

## Características

El conversor binario de Visual J# admite:

- Conversión de archivos .class generados desde determinado código de Java. El Conversor binario de Visual J# convertirá el código que utiliza la funcionalidad de la mayoría de las bibliotecas de clases de JDK nivel 1.1.4.
- Conversión de archivos .class que contengan la mayoría de las extensiones de Microsoft Visual J++® 6.0, incluidos los delegados y J/Direct®.

El conversor binario de Visual J# proporciona una serie de opciones que permiten al usuario realizar las siguientes acciones:

- Especificar un ensamblado DLL o EXE.
- Crear un ensamblado con nombre seguro que se pueda instalar en la Caché de ensamblados global (GAC).
- Importar y utilizar código de ensamblado de .NET Framework existente.
- Buscar archivos .class de forma recursiva en subdirectorios para convertirlos.

JbImp.exe no admite:

- Código que utiliza tecnología de interoperabilidad Java y COM.
- Código que utiliza funcionalidad de las bibliotecas de clases por encima del nivel 1.1.4 de JDK. JbImp.exe no admite código que utilice determinada funcionalidad de las bibliotecas de clases de JDK nivel 1.1.4, incluidos RMI, RNI, JNI y subprogramas.

## Sintaxis y opciones

La sintaxis y las opciones de la herramienta son:

```
JbImp [options] <class_files> [ [options] <class_files> ...]
```

Argumento	Descripción
archivos_class	Nombres de los archivos .class que se van a convertir. En esta opción se enumeran directorios y archivos. Los nombres de directorio y archivo pueden contener asteriscos (*) y signos de interrogación (?) como caracteres comodín. También se puede especificar un archivo CAB, ZIP o JAR.
Opción	Descripción
<b>/delaysign</b>	Especifica si un ensamblado estará firmado total o parcialmente. Utilice <b>/delaysign</b> si desea incluir sólo una clave pública en el ensamblado.  De manera predeterminada, <b>/delaysign</b> no está activado.  La opción <b>/delaysign</b> no tiene ningún efecto a menos que se utilice con <b>/keyfile</b> o <b>/keycontainer</b> .
<b>/help</b> <b>/?</b>	Imprime un resumen de las opciones de JbImp.exe.
<b>/k[keyfile]:archivo</b>	Firma el ensamblado generado con el archivo de claves. Se crea un ensamblado con nombre seguro con el par de claves especificado en <i>archivo</i> . Este archivo se suele generar con la utilidad sn.exe. Si la cadena contiene algún espacio, hay que escribirla entre comillas dobles (" ").



<b>/keycontainer:</b> <i>cadena</i>	Especifica un nombre de contenedor de claves para un par de claves con el fin de asignar un nombre seguro a un ensamblado. Si la cadena contiene algún espacio, hay que escribirla entre comillas dobles (" ").
<b>/linkres[ource]:</b> <i>archivo[,identificador]</i>	Crea un vínculo a un recurso administrado. Opcionalmente, especifica un nombre lógico que se utiliza para cargar el recurso. El valor predeterminado es el nombre del archivo.
<b>/m[ain]:</b> <i>clase</i>	<p>Especifica el punto de entrada para el archivo ejecutable. La lista de archivos .class se analiza para encontrar la clase con el nombre <i>clase</i>. Si esta clase existe y tiene un método con la firma <code>public static void main(String[] args)</code>, este método se define como el punto de entrada. De lo contrario, se detiene la conversión.</p> <p>Si no se selecciona esta opción, la primera clase que tenga un método con la firma <code>public static void main(String[] args)</code> será el punto de entrada. Si no existe esta clase y se selecciona la opción <b>/target:exe</b>, la conversión se detiene.</p> <p>El nombre de clase especificado en el argumento puede ser un nombre sencillo o un nombre completo que incluya el nombre del paquete.</p>
<b>/nologo</b>	Suprime la información de pancarta del conversor.
<b>/out:</b> <i>archivo</i>	Especifica el nombre del archivo de resultados y crea un archivo de ensamblado de .NET Framework con el nombre <i>archivo</i> .
<b>/r[eference]:</b> <i>archivos</i>	Utiliza los <i>archivos</i> de ensamblado del Lenguaje intermedio de Microsoft (MSIL) de .NET Framework para resolver referencias a metadatos en los archivos .class.
<b>/recurse</b> <i>dir</i>	Busca archivos .class en los subdirectorios del directorio principal <i>dir</i> . Esta opción es efectiva sólo cuando <i>archivos_class</i> incluye nombres de directorio.
<b>/linkres[ource]:</b> <i>archivo[,identificador]</i>	Incrusta un recurso administrado en el ensamblado. Opcionalmente, especifica un nombre lógico que se utiliza para cargar el recurso. El valor predeterminado es el nombre del archivo.
<b>/securescoping</b>	Sigue las reglas de ámbito seguro. Ésta es una opción avanzada que se puede utilizar para asignar un ámbito de paquete de Java a un ámbito de ensamblado de .NET Framework. Si no se especifica esta opción, los miembros con ámbito protegido y de paquete del código de bytes de Java se asignan a un ámbito público en el ensamblado de salida.
<b>/t[arget]:library</b>	Genera una DLL. No define ningún punto de entrada, ni siquiera si está presente <code>main</code> .
<b>/t[arget]:exe</b>	Genera un archivo ejecutable (.exe). Éste es el valor predeterminado para la opción <b>/target</b> . Uno de los archivos .class de entrada debe implementar <code>main</code> .
<b>/t[arget]:module</b>	Genera un módulo MSIL. Common Language Runtime no puede cargar ni ejecutar un módulo, pero el compilador de Visual J# y otros compiladores pueden utilizar un módulo para tener acceso a los tipos del mismo.
<b>/usestubs</b>	<p>Utiliza tipos stub, campos y métodos para las referencias no resueltas.</p> <p>Si se hace referencia a una clase y no está presente en la entrada, se busca automáticamente la variable de entorno CLASSPATH para la clase que falta y la primera ubicación encontrada se muestra con la información de error. Si no se especifica esta opción, se detiene la conversión si alguna de las clases a las que se hace referencia no están presentes en la entrada. Si se especifica esta opción, los tipos stub se emiten en el ensamblado creado para cada una de las clases que faltan. Esto hará que la conversión sea correcta, pero dará lugar a que se inicie una excepción cuando se intente tener acceso a los miembros de una clase que falta en tiempo de ejecución.</p>

## Vea también

[Herramientas de Visual J#](#) | [Referencia](#) | [Opciones del compilador de Visual J#](#)

# Referencia de la interfaz de usuario

En esta sección se proporciona información de referencia sobre las características del entorno de desarrollo de Visual Studio® .NET que son exclusivas de Visual J#.

## En esta sección

### [Editor de texto, Opciones \(Cuadro de diálogo\)](#)

Proporciona vínculos a temas de referencia sobre las páginas de la carpeta del Editor de texto del cuadro de diálogo Opciones que son exclusivas de Visual J#.

### [Propiedades de un proyecto de Visual J#](#)

Proporciona vínculos a temas de referencia sobre las propiedades de los proyectos de Visual J#.

## Secciones relacionadas

### [Establecer las propiedades de un proyecto de Visual J#](#)

Proporciona instrucciones para tener acceso a propiedades de proyecto.

### [Asistente para actualización a Visual J#](#)

Proporciona información general sobre el Asistente para actualización a Visual J# e información detallada sobre el trabajo con sus características.

### [Comentarios sobre documentación en Visual J#](#)

Explica las características del entorno de desarrollo integrado que proporcionan compatibilidad con comentarios sobre documentación.

### [Referencia](#)

Temas de referencia sobre el compilador de Visual J#, el Conversor binario (Jblmp.exe), la compatibilidad de bibliotecas y lenguajes, la actualización de proyectos de Visual J++ 6.0, la sintaxis de Visual J# para la biblioteca de clases de .NET Framework y las características del entorno de desarrollo.

# Editor de texto, Opciones (Cuadro de diálogo)

La carpeta Editor de texto del cuadro de diálogo Opciones permite cambiar la configuración predeterminada del Editor de código y de texto. Las páginas siguientes son específicas para Visual J#:

- [General, Visual J#, Editor de texto, Opciones \(Cuadro de diálogo\)](#)
- [Tabulaciones, Visual J#, Editor de texto, Opciones \(Cuadro de diálogo\)](#)
- [Formato, Visual J#, Editor de texto, Opciones \(Cuadro de diálogo\)](#)

## Vea también

[Referencia de la interfaz de usuario | Opciones \(Cuadro de diálogo\)](#)

# General, Visual J#, Editor de texto, Opciones (Cuadro de diálogo)

La página de propiedades **General**, en la carpeta **Visual J#** de la carpeta **Editor de texto** del cuadro de diálogo **Opciones** (menú **Herramientas**), contiene las siguientes propiedades:

## Finalización de instrucciones

### Lista de miembros automática

Al seleccionar esta opción, aparece una lista de miembros disponibles de un objeto o clase dados en los lugares correspondientes al escribir una línea de código. Vea [Lista de miembros](#) para obtener más información.

### Ocultar miembros avanzados

Al seleccionar esta opción, se ocultan ciertos miembros marcados internamente como "avanzados" en la lista de finalización de instrucciones. Los miembros avanzados son aquellos que existen sólo por motivos de infraestructura, pero que deben poderse ver. Esta opción permite filtrarlos para que no sean visibles si no es necesario verlos.

### Información de parámetros

Al seleccionar esta opción, se muestra la información de parámetros de la función o procedimiento, si la hay. Vea [Información de parámetros](#) para obtener más información.

## Configuración

### Habilitar espacio virtual

Al seleccionar esta opción, es posible mover el cursor más allá del extremo físico de una línea de texto, hacia espacios de carácter que no existen. Si mueve el cursor en esta área y comienza a escribir, el espacio existente entre su posición actual y el final de la última palabra se llena automáticamente con caracteres de espacio. Cuando no está activada esta opción, no se puede mover el cursor más allá del último carácter de una línea.

### Ajuste de línea

Al seleccionar esta opción, las líneas que se extienden horizontalmente más allá del intervalo visible del Editor de texto se ajustan automáticamente a la línea siguiente.

## Presentación

### Números de línea

Al seleccionar esta opción, aparece un número de línea a la izquierda de cada línea de texto.

### Habilitar acceso a direcciones URL con un solo clic

Al seleccionar esta opción, las direcciones URL incrustadas en el texto requieren únicamente un clic del *mouse* (ratón) para ir a la página Web correspondiente.

### Barra de exploración

Al seleccionar esta opción, se muestra la barra de exploración, que se encuentra en la parte superior del Editor de código. Esto permite explorar un objeto o procedimiento particular dentro de ese objeto.

## Vea también

[Editor de texto, Opciones \(Cuadro de diálogo\)](#) | [General, Entorno, Opciones \(Cuadro de diálogo\)](#)

# Tabulaciones, Visual J#, Editor de texto, Opciones (Cuadro de diálogo)

La página de propiedades **Tabulaciones**, en la carpeta **Visual J#** de la carpeta **Editor de texto** del cuadro de diálogo **Opciones** (menú **Herramientas**) permite cambiar la configuración de tabulación predeterminada del Editor de código y de texto.

## Sangría

### Ninguna

Al seleccionar esta opción, no se aplica ninguna sangría de forma automática al presionar ENTRAR para desplazarse a una nueva línea de texto. El cursor se coloca en la primera columna de la línea siguiente.

### Bloque

Al seleccionar esta opción y presionar ENTRAR, se aplica de forma automática una sangría a la nueva línea de texto hasta la misma posición de tabulación que la línea precedente.

### Automática

Al seleccionar esta opción, las nuevas líneas de texto reciben formato automático según lo establecido en el servicio de lenguaje actual. Por ejemplo, si se escribe un bucle **for**, se aplicará una sangría a la línea siguiente de forma automática hasta la siguiente posición de tabulación a la derecha.

## Fichas

### Tamaño de tabulación

Especifica el número de espacios que representa un carácter de tabulación.

### Tamaño de sangría

Especifica el número de espacios que deben insertarse al aplicar una sangría cada vez que se presiona la tecla TABULADOR. El espacio insertado puede ser una combinación de caracteres de tabulación o de espacio.

### Insertar espacios

Al seleccionar esta opción, las operaciones de sangría insertan caracteres de espacio en lugar de caracteres de tabulación. Si el tamaño de sangría es 5, por ejemplo, se insertan cinco espacios en la línea actual de texto cada vez que se presiona la tecla TABULADOR o el botón Sangría.

### Mantener tabulaciones

Al seleccionar esta opción, las operaciones de sangría insertan caracteres de tabulación en lugar de espacios siempre que sea posible. El carácter de tabulación llena el número de espacios especificado en **Tamaño de tabulación**. Si el tamaño de sangría no es un múltiplo par del tamaño de tabulación, se insertan caracteres de espacio para llenar el espacio restante.

## Vea también

[Editor de texto, Opciones \(Cuadro de diálogo\)](#) | [General, Entorno, Opciones \(Cuadro de diálogo\)](#)

# Formato, Visual J#, Editor de texto, Opciones (Cuadro de diálogo)

La página de propiedades **Formato**, en la carpeta **Visual J#** de la carpeta **Editor de texto** del cuadro de diálogo **Opciones** (menú **Herramientas**), contiene las siguientes propiedades:

## Mantener llaves de apertura en la misma línea que la construcción

El Editor de código desplaza la llave de apertura de una construcción, como una instrucción **if**, hasta la línea situada tras la declaración de la instrucción, alineando la llave de apertura con el primer carácter de la declaración. Cuando esta opción está seleccionada, el editor no desplaza una llave de apertura desde la línea de declaración.

## Aplicar sangría a etiquetas de caso

Una instrucción **case** se alinea con el primer carácter de una instrucción **switch**. Cuando esta opción está seleccionada, el Editor de código aplica sangría a las instrucciones **case**.

## Aplicar formato automáticamente a las construcciones finalizadas y al código fuente pegado

De manera predeterminada, cuando se escriben construcciones de código, el Editor de código puede aplicarles formato y alinearlas. Asimismo, cuando se pega código, el editor puede aplicar formato al texto. Si se selecciona esta opción, el editor alinea el texto que escribe y pega, y le da formato.

## Edición de comentarios automática

Cuando se utilizan comentarios de código Javadoc, el Editor de código puede completar el comentario después de escribir la introducción de comentario `/**`. Cuando se activa esta opción, el editor inserta los caracteres `*/` en la línea siguiente para cerrar el comentario.

## Especificar el modo de esquematización al abrir los archivos

Cuando se coloca un archivo en el Editor de código, se puede habilitar la función de esquematización. Vea [Esquematizar y ocultar código](#) para obtener más información. Cuando esta opción está seleccionada, la función de esquematización se habilita al abrir un archivo.

## Contraer los bloques `#region` al abrir el archivo

Si en el código hay uno o varios bloques `#region`, se puede especificar que los bloques estén abiertos o cerrados al abrir el archivo. Cuando esta opción está seleccionada, los bloques `#region` están contraídos cuando se abre el archivo.

## IntelliSense preselecciona los miembros usados con mayor frecuencia

La tecnología IntelliSense está diseñada para preseleccionar a los miembros que se utilizan con más frecuencia al autocompletar nombres de objetos. Para obtener más información, vea [IntelliSense para los miembros utilizados con mayor frecuencia](#).

Cuando esta opción está activada, IntelliSense preseleccionará el miembro que utilice con mayor frecuencia en el cuadro emergente [Lista de miembros](#).

## Borrar historial

Borra la lista de miembros que se utiliza al autocompletar los nombres de objetos.

## Vea también

[Editor de texto, Opciones \(Cuadro de diálogo\)](#) | [General](#), [Entorno](#), [Opciones \(Cuadro de diálogo\)](#)

# Propiedades de un proyecto de Visual J#

Para obtener información sobre cómo configurar las propiedades de los proyectos de Visual J#, vea los siguientes temas.

- [General, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Configuración del Web, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Valores predeterminados del diseñador, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Ruta de acceso de referencias, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Generar eventos, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Generar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Depurar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)
- [Avanzadas, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#)

## Vea también

[Establecer las propiedades de un proyecto de Visual J# | Referencia de la interfaz de usuario](#)

# General, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **General** de la carpeta **Propiedades comunes** contiene las siguientes propiedades:

## Aplicación

### Nombre del ensamblado

Nombre del archivo de resultados que contiene el manifiesto del ensamblado. Si se cambia esta propiedad, también se cambia la propiedad **Archivo de resultados**.

Vea [AssemblyName \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Tipo de resultado

Indica el tipo de aplicación que se va a generar. En un proyecto de aplicación Web, esta propiedad sólo se puede establecer en Biblioteca de clases.

Vea [OutputType \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Paquete predeterminado

Paquete correspondiente a aquellos elementos que, como las clases, se agregan a través del [cuadro de diálogo Agregar nuevo elemento](#).

### Objeto inicial

Nombre de la clase que contiene el método main al que se desea llamar al comienzo del programa. Si selecciona **Sin establecer**, se utilizará el primer archivo encontrado que contenga un método main.

Vea [/main](#) y [StartupObject \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Motores en tiempo de ejecución compatibles

Especifica la versión de Common Language Runtime admitida por la aplicación. A menos que se indique lo contrario, la aplicación se ejecutará con la versión de motor de tiempo de ejecución que se utiliza para su generación. Si desea que la aplicación se ejecute en cualquier versión de Common Language Runtime, debe modificar el valor de esta propiedad.

Esta propiedad no se puede utilizar en el código. Sólo está disponible para proyectos de Aplicación para Windows y proyectos de Aplicación Web ASP.NET. En una aplicación para Windows, cuando se selecciona un motor de tiempo de ejecución, el sistema del proyecto actualiza el archivo app.config del proyecto o, si este archivo no existe, crea un archivo app.config. En tiempo de ejecución, el nombre del archivo app.config debe ser *nombre\_archivo.ext.config* (donde *nombre\_archivo.ext* es el nombre del ejecutable que inició la aplicación) y el archivo debe estar en el mismo directorio que el ejecutable. El sistema del proyecto creará el archivo *nombre\_archivo.ext.config* a partir del archivo app.config y lo colocará en el directorio bin\target.

Si especifica más de una versión de motor de tiempo de ejecución y la aplicación de Windows se ejecuta en un equipo con más de una versión del motor de tiempo de ejecución instalada, se utilizará la primera versión especificada en el archivo .config que coincida con un motor de tiempo de ejecución instalado que esté disponible.

Para obtener más información, vea [Especificar una versión de .NET Framework](#).

En proyectos de Aplicación Web ASP.NET, cuando se modifica el valor de esta propiedad, el sistema del proyecto actualiza el archivo web.config del proyecto agregando etiquetas de enlace de ensamblado (<bindingRedirect> y <assemblyIdentity>) en una sección <dependentAssembly>. Estas instrucciones de enlace garantizan que se utilizará un conjunto coherente de ensamblados de .NET Framework para la aplicación cuando se esté ejecutando con una versión más antigua del motor de tiempo de ejecución. El archivo web.config permanecerá en el directorio del proyecto y su nombre no cambiará.

Al implementar la aplicación en un entorno con más de una versión del motor de tiempo de ejecución, debe asegurarse de que el directorio virtual de la aplicación en IIS se asigna mediante una secuencia de comandos a la versión deseada de ASP.NET. Es posible asignar una aplicación mediante una secuencia de comandos a una versión de ASP.NET con la herramienta de línea de comandos aspnet\_regiis.exe asociada a la versión deseada. Vea la documentación de aspnet\_regiis.exe para obtener más información.

## Proyecto

### Archivo de proyecto



Nombre del archivo correspondiente al proyecto actual. Es una propiedad de sólo lectura mientras el proyecto está abierto en Visual Studio.

Vea [FileName \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### **Carpeta de proyecto**

Ruta de acceso al directorio del proyecto. Mientras el proyecto esté abierto en Visual Studio esta propiedad es de sólo lectura.

Vea [LocalPath \(Propiedad\)](#) y [FullPath \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### **Archivo de resultados**

Nombre del archivo de resultados principal del proyecto (también conocido como imagen del programa). El archivo que contiene los metadatos de ensamblado debe ser el archivo de resultados de esta aplicación. Por ello, **Archivo de resultados** es una propiedad de sólo lectura que sólo puede modificarse cambiando la propiedad **Nombre del ensamblado**.

Vea [OutputFileName \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## **Ensamblado de contenedor para objetos ActiveX y COM**

### **Archivo de clave de ensamblado de contenedor**

Archivo de clave con el que se firman los componentes COM a los que hace referencia este proyecto. Se puede hacer referencia a los componentes COM a través del cuadro de diálogo [Agregar referencia](#). Utilice sn.exe para generar un par de claves y, a continuación, registrarlo en un contenedor.

Vea [AssemblyOriginatorKeyFile \(Propiedad\)](#) y [AssemblyOriginatorKeyMode \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### **Nombre de la clave de ensamblado del contenedor**

Contenedor de claves con el que se firman los componentes COM a los que hace referencia este proyecto. Se puede hacer referencia a los componentes COM a través del cuadro de diálogo [Agregar referencia](#). Utilice sn.exe para generar un contenedor de claves.

El contenedor no tiene por qué almacenarse en el equipo local; puede hallarse en un dispositivo externo conectado al equipo.

Vea [AssemblyKeyContainerName \(Propiedad\)](#) y [AssemblyOriginatorKeyMode \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

Para obtener información sobre cómo tener acceso a la página de propiedades **General** de la carpeta **Propiedades comunes**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

### **Vea también**

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Configuración del Web, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Configuración del Web** de la carpeta **Propiedades comunes**, que sólo está disponible para los proyectos de Aplicación Web, contiene las propiedades siguientes:

## Conexión con el servidor Web

### Modo de acceso al Web

Especifica el método de acceso a los archivos del proyecto en el servidor. Esta propiedad permite reemplazar la configuración predeterminada del equipo, que se especifica en el cuadro de diálogo **Opciones** (menú **Herramientas**), carpeta **Proyectos**, página **Configuración del Web**. El valor de esta propiedad sólo surte efecto cuando se abre el proyecto.

El valor predeterminado de esta propiedad se establece en [Configuración del Web, Proyectos, Opciones \(Cuadro de diálogo\)](#).

Vea [WebAccessMethod \(Propiedad\)](#) para obtener más información.

### Reparación de vínculos

Cuando la propiedad **Modo de acceso al Web** está establecida en FrontPage, esta opción especifica si está habilitada la reparación de vínculos de las Extensiones de servidor de FrontPage. La reparación de vínculos actualiza aquellos vínculos a páginas existentes en el proyecto en caso de que cambien durante el desarrollo del mismo. Los cambios efectuados en esta propiedad sólo son válidos cuando la propiedad **Modo de acceso al Web** se establece en FrontPage.

Vea [LinkRepair \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Dirección URL sin conexión

Especifica el directorio raíz del proyecto Web cuando se trabaja sin conexión. Los archivos del proyecto se encuentran en el servidor (especificado al crear el proyecto) y en una dirección URL sin conexión. Cuando se trabaja con el proyecto sin estar conectado (menú **Proyecto**, opción de menú **Proyecto Web**), no se realiza ningún intento de escribir ni de utilizar los archivos del proyecto en el servidor. Se permite cualquier directorio válido.

El valor predeterminado de esta propiedad se establece en [Configuración del Web, Proyectos, Opciones \(Cuadro de diálogo\)](#).

Vea [OfflineURL \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Ruta de acceso

Se trata de la ruta de acceso al directorio del proyecto en el servidor. Esta propiedad únicamente mostrará un valor cuando se establezca la propiedad **Modo de acceso al Web** en Recurso compartido de archivos. El valor de esta propiedad se utiliza cuando se abre el proyecto en Visual Studio.

Vea [FileSharePath \(Propiedad\)](#) y [ActiveFileSharePath \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Información del servidor Web

### Servidor Web

Muestra el nombre del servidor Web de este proyecto. Se especifica el servidor al crear el proyecto.

Vea [WebServer \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Versión del servidor Web

Muestra la versión de Internet Information Server (IIS) existente en el servidor especificado en la propiedad **Servidor Web**.

Vea [WebServerVersion \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Versión de las Extensiones de servidor

Cuando la propiedad **Modo de acceso al Web** se ha establecido en FrontPage, esta propiedad muestra la versión de las Extensiones de FrontPage instaladas en el servidor.

Vea [ServerExtensionsVersion \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

Para obtener información sobre cómo obtener acceso a la página de propiedades **Configuración del Web** de la carpeta **Propiedades comunes**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

#### **Vea también**

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Valores predeterminados del diseñador, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Valores predeterminados del diseñador** de la carpeta **Propiedades comunes**, que se utiliza para especificar el diseño predeterminado y los valores de configuración de los diseñadores de formularios Web Forms, contiene las siguientes propiedades:

## Diseñadores Web

### Diseño de página

Especifica si las páginas de formularios Web Forms agregadas al proyecto deben presentar un formato lineal (unidimensional) o de cuadrícula (bidimensional) de forma predeterminada. Esta propiedad de proyecto puede reemplazarse en un archivo a través de la [ventana Propiedades](#).

Vea [DefaultHTMLPageLayout \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Esquema de destino

Especifica la plataforma HTML con la que se desea garantizar la compatibilidad de la aplicación.

Esta propiedad de proyecto puede reemplazarse en un archivo a través de la ventana Propiedades. Para obtener acceso a esta propiedad mediante programación, vea [DefaultTargetSchema \(Propiedad\)](#). Para obtener más información sobre el esquema de destino, vea [Establecer la propiedad targetSchema de un documento HTML](#).

## Secuencia de comandos Web

### Lenguaje de secuencias de comandos del cliente

Especifica el lenguaje de secuencias de comandos que se va a utilizar al generar código que se ejecuta en el cliente. Esta propiedad de proyecto puede reemplazarse en un archivo mediante la [ventana Propiedades](#).

Vea [DefaultClientScript \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

Para obtener información sobre cómo tener acceso a la página de propiedades **Valores predeterminados del diseñador** de la carpeta **Propiedades comunes**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

## Vea también

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Ruta de acceso de referencias, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Ruta de acceso de referencias** de la carpeta **Propiedades comunes** contiene las siguientes propiedades:

## Ruta de acceso de referencias

Enumera los directorios que contienen ensamblados examinados en el cuadro de diálogo **Agregar referencia**.

La configuración de esta propiedad es específica del proyecto, equipo y usuario para los cuales se estableció y su información no se almacenará en la configuración del proyecto.

Vea [Agregar y quitar referencias](#) y [ReferencePath \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

Cuando el sistema correspondiente al proyecto encuentra una referencia de ensamblado, intenta resolver la referencia buscando en las siguientes ubicaciones, por el siguiente orden:

1. Se buscan ensamblados en el directorio del proyecto. Un directorio es directorio de proyecto si los archivos que contiene se muestran en el Explorador de soluciones. Esto no incluye los archivos que se muestran cuando la opción **Mostrar todos los archivos** se encuentra activada.
2. Se buscan ensamblados en los directorios especificados en la propiedad **Ruta de acceso de referencia**.
3. Se buscan ensamblados en los directorios utilizados para mostrar archivos en la ficha .NET del [cuadro de diálogo AgregarReferencia](#).
4. Se buscan ensamblados en el directorio obj del proyecto; cualquier ensamblado creado como resultado de agregar una referencia COM al proyecto se agregará aquí.

Para obtener información sobre cómo tener acceso a la página de propiedades **Ruta de acceso de referencias** de la carpeta **Propiedades comunes**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

## Vea también

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Generar eventos, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Eventos de generación** de la carpeta **Propiedades comunes** contiene las siguientes propiedades:

**Nota** No está disponible para aplicaciones Web.

## Línea de comandos de evento anterior a la generación

Comando que se va a ejecutar antes de que se inicie la generación. Para escribir comandos largos, haga clic en el botón de puntos suspensivos (...) situado al final de la línea para mostrar un cuadro de edición.

## Línea de comandos de evento posterior a la generación

Comando que se va a ejecutar cuando haya finalizado la generación. Para escribir comandos largos, haga clic en el botón de puntos suspensivos situado al final de la línea para mostrar un cuadro de edición.

## ¿Desea ejecutar el evento posterior a la generación?

Especifica las siguientes condiciones para que se ejecute el evento posterior a la generación, como se indica en la tabla siguiente.

Opción	Resultado
<b>Siempre</b>	El evento posterior a la generación se ejecutará independientemente de si la generación finaliza correctamente.
<b>Si la generación es correcta</b>	El evento posterior a la generación se ejecutará si la generación finaliza correctamente. Así, el evento se ejecutará incluso en un proyecto actualizado, con tal de que la generación finalice correctamente.
<b>Cuando la generación actualiza los resultados del proyecto</b>	El evento posterior a la generación sólo se ejecutará cuando el archivo de resultados del compilador (.exe o .dll) difiera del anterior archivo de resultados del compilador. Así, un evento posterior a la generación no se ejecutará si un proyecto está actualizado.

Los eventos de generación se pueden utilizar para especificar comandos que se ejecuten antes de iniciarse la generación o al terminarse la generación. Los eventos de generación sólo se ejecutarán si se han alcanzado correctamente dichos puntos en el proceso de generación.

Cuando se genera un proyecto, los eventos anteriores a la generación se incluyen en un archivo denominado PreBuildEvent.bat y los eventos posteriores a la generación se incluyen en un archivo denominado PostBuildEvent.bat.

**Nota** Si desea garantizar la comprobación de errores, agregue sus propios comandos de comprobación de errores a los pasos de generación.

## Para especificar un evento de generación

1. En el Explorador de soluciones, seleccione el proyecto para el que desee especificar el evento de generación.
2. Abra el cuadro de diálogo **Páginas de propiedades** del proyecto. Para obtener más información, vea [Establecer las propiedades de un proyecto de Visual J#](#).
3. En la carpeta **Eventos de generación**, seleccione un evento de generación y especifique la sintaxis del evento de generación.

La sintaxis puede incluir cualquier comando que sea válido en la línea de comandos o en un archivo .bat. El nombre de los archivos de proceso por lotes deberá ir precedido por **call** para garantizar la ejecución de todos los comandos posteriores.

Es posible que desee utilizar las macros siguientes para especificar ubicaciones de archivos u obtener el nombre real del archivo de entrada en el caso de múltiples selecciones. Estas macros no hacen distinción entre mayúsculas y minúsculas.

Macro	Descripción
<b>\$(ConfigurationName)</b>	Nombre de la configuración del proyecto actual (por ejemplo, "Versión de depuración").
<b>\$(PlatformName)</b>	Nombre de la plataforma del proyecto actual, por ejemplo, "Win32".
<b>\$(OutDir)</b>	Ruta de acceso (relativa al directorio del proyecto) al directorio de archivos de resultados. Se resuelve en el valor de la propiedad <b>Directorio de resultados</b> . Incluye la barra diagonal inversa final '\'.
<b>\$(DevEnvDir)</b>	Directorio de instalación de Visual Studio.NET (definido como unidad + ruta de acceso); incluye una barra diagonal inversa ('\') final.

<b>\$(ProjectDir)</b>	Directorio del proyecto (definido como unidad + ruta de acceso); incluye una barra diagonal inversa ('\') final.
<b>\$(ProjectPath)</b>	Nombre de la ruta de acceso absoluta del proyecto (definido como unidad + ruta de acceso + nombre base + extensión de archivo).
<b>\$(ProjectName)</b>	Nombre base del proyecto.
<b>\$(ProjectFileName)</b>	Nombre de archivo del proyecto (definido como nombre base + extensión de archivo).
<b>\$(ProjectExt)</b>	Extensión de archivo del proyecto. Incluye '.' antes de la extensión del nombre de archivo.
<b>\$(SolutionDir)</b>	Directorio de la solución (definido como unidad + ruta de acceso); incluye una barra diagonal inversa ('\') final.
<b>\$(SolutionPath)</b>	Nombre de la ruta de acceso absoluta de la solución (definido como unidad + ruta de acceso + nombre base + extensión de archivo).
<b>\$(SolutionName)</b>	Nombre base de la solución.
<b>\$(SolutionFileName)</b>	Nombre de archivo de la solución (definido como nombre base + extensión de archivo).
<b>\$(SolutionExt)</b>	Extensión de archivo de la solución. Incluye '.' antes de la extensión del nombre de archivo.
<b>\$(TargetDir)</b>	Directorio del archivo de resultados principal de la generación (definido como unidad + ruta de acceso). Incluye la barra diagonal inversa final '\'.
<b>\$(TargetPath)</b>	Nombre de la ruta de acceso absoluta del archivo de resultados principal de la generación (definido como unidad + ruta de acceso + nombre base + extensión de archivo).
<b>\$(TargetName)</b>	Nombre base del archivo de resultados principal de la generación.
<b>\$(TargetFileName)</b>	Nombre del archivo de resultados principal de la generación (definido como nombre base + extensión de archivo).
<b>\$(TargetExt)</b>	Extensión del archivo de resultados principal de la generación. Incluye '.' antes de la extensión del nombre de archivo.

Vea [PreBuildEvent](#), [PostBuildEvent](#) y [RunPostBuildEvent](#) para obtener información sobre como tener acceso mediante programación al valor de esta propiedad.

## Vea también

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Generar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Generar** de la carpeta **Propiedades de configuración** contiene las siguientes propiedades:

## Generación de código

### Constantes de compilación condicional

Especifica los símbolos en los que se llevará a cabo la compilación condicional. Separe los símbolos con un espacio. Vea [/define](#) para obtener más información.

Vea [DefineConstants \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Optimizar código

Habilita o deshabilita las optimizaciones realizadas por el compilador para hacer que el archivo de resultados sea más pequeño, rápido y eficiente.

Vea [Optimize \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Errores y advertencias

### Nivel de advertencia

Especifica el nivel de advertencia que deberá mostrar el compilador. Vea [/warn](#) para obtener más información.

Vea [WarningLevel \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Tratar advertencias como errores

Trata todas las advertencias como errores. Vea [/warnaserror](#) para obtener más información.

Vea [TreatWarningsAsErrors \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Suprimir advertencias específicas

Bloquea la capacidad del compilador de generar una o más advertencias. Si hay varios números de advertencia, hay que separarlos con una coma. Para obtener más información, vea [/nowarn](#).

Vea [WarningLevel \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Resultados

### Ruta de acceso de los resultados

Especifica la ubicación de los archivos de resultados para esta configuración del proyecto.

Vea [OutputPath \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Generar información de depuración

Genera información de depuración y la coloca en el archivo de resultados. Vea [/debug](#) para obtener más información.

Vea [DebugSymbols \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Registrar para interoperabilidad COM

Indica que nuestra aplicación administrada expondrá un objeto COM (un contenedor al que se puede llamar mediante COM) que permite a un objeto COM interactuar con la aplicación administrada. La propiedad **Tipo de resultados** de esta aplicación debe ser **Biblioteca de clases** para que la propiedad **Registrar para interoperabilidad COM** esté disponible. Para obtener más información, vea [General, Propiedades comunes, Páginas de propiedades de <Nombre de proyecto> \(Cuadro de diálogo\)](#).

Para obtener más información sobre cómo obtener acceso mediante programación a la propiedad **Registrar para interoperabilidad COM**, vea [RegisterForComInterop \(Propiedad\)](#).

Para obtener información sobre cómo tener acceso a la página de propiedades **Generar** de la carpeta **Propiedades de configuración**, vea [Establecer las propiedades de un proyecto de Visual J#](#).



**Vea también**

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Depurar, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Depurar** de la carpeta **Propiedades de configuración** contiene las siguientes propiedades:

## Depuradores

### Habilitar depuración ASP

Habilita la depuración de páginas Active Server (ASP).

Vea [EnableASPDebugging \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Habilitar depuración ASP.NET

Habilita la depuración de las aplicaciones escritas para la plataforma de desarrollo Web ASP.NET.

El atributo de depuración de la ficha <compilation> del archivo web.config del proyecto especifica si se generarán símbolos de depuración (archivos .pdb) para los archivos ASP.NET compilados dinámicamente, como los de extensión .aspx, .asmx y .ascx.

Debido a que numerosas partes de una aplicación ASP.NET se compilan dinámicamente en tiempo de ejecución, como por ejemplo los archivos .aspx y .ascx, se debe configurar el motor en tiempo de ejecución de ASP.NET para que compile la aplicación con información de depuración con el fin de poder depurar los archivos.

Para insertar información de depuración en la página compilada, establezca debug=true.

Al establecer debug=true se crea un archivo de mayor tamaño. Por tanto, este valor sólo debe habilitarse al depurar una aplicación, porque puede afectar significativamente a su rendimiento.

Después de cambiar el valor de esta propiedad, seleccione el botón **Aplicar** para habilitar la edición de la propiedad apropiada (**Aplicación de inicio**, **Dirección URL de inicio** o **Página de inicio**).

Vea [EnableASPXDebugging \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Habilitar depuración sin administrar

Habilita la depuración de código Win32 nativo (sin administrar), al cual se llama desde la aplicación de Visual J#.

Vea [EnableUnmanagedDebugging \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Habilitar depuración de SQL Server

Habilita la depuración de procedimientos SQL desde la aplicación de Visual J#.

Vea [EnableSQLServerDebugging \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Acción de inicio

### Modo de depuración

Especifica si se desea depurar un proyecto, programa o dirección URL. También se puede seleccionar **Espere a conectarse a un proceso externo**.

- **Proyecto** Indica que el ejecutable (para los proyectos de Aplicación para Windows y Aplicación de consola) o la página de inicio (para proyectos Web) deben iniciarse cuando se depure la aplicación. Los proyectos de biblioteca de clases no pueden iniciarse directamente. Si se desea depurar un Servicio Web, establecer esta propiedad hará que la aplicación quede asociada automáticamente a Internet Explorer.
- **Programa** Indica que debe iniciarse un programa específico cuando se depure la aplicación. Puede ejecutar, por ejemplo, un cliente previamente generado que utilice el proyecto de Biblioteca de clases. Se debe establecer una **Aplicación de inicio** para poder depurar.
- **Dirección URL** Indica que se debe tener acceso a una dirección URL (**Dirección URL de inicio**) específica cuando se depure la aplicación. Es posible obtener acceso, por ejemplo, a la dirección URL de un sitio Web en el que se utiliza el proyecto de biblioteca de clases.
- **Espere a conectarse a un proceso externo** Este valor sólo se aplica a los proyectos de Aplicación Web y Servicio Web XML. Para un proyecto de servicio Web XML, el depurador se conectará a un proceso que llamará a su vez al servicio Web

XML. Establezca un punto de interrupción en el Servicio Web XML antes de comenzar la depuración, para que se alcance el punto cuando el proceso externo llame al servicio.

Vea [StartAction \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Aplicación de inicio

Especifica el comando usado para iniciar el programa que se está depurando.

Vea [StartProgram \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Dirección URL de inicio

Si se establece el **Modo de depuración** en Dirección URL, especifica la dirección URL donde se encuentra el proyecto que está depurando.

Vea [StartURL \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Página de inicio

Especifica la dirección URL de la página que se debe utilizar como página de inicio al depurar.

Vea [StartPage \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

### Opciones de inicio

#### Argumentos de la línea de comandos

Especifica los argumentos de la propiedad **Aplicación de inicio**.

Vea [CommandLineArguments \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

#### Directorio de trabajo

Especifica el directorio de trabajo del programa que se depura.

Vea [StartWorkingDirectory \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

#### Siempre usar Internet Explorer

Determina si la depuración usará el explorador de Internet predeterminado para Visual Studio.

Vea [StartWithIE \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

#### Habilitar depuración remota

Al activar esta opción, permite la depuración remota de un archivo .exe en el equipo especificado en la propiedad **Equipo de depuración remoto**.

Para obtener acceso a esta propiedad mediante programación, vea [RemoteDebugEnabled](#).

#### Equipo de depuración remoto

Quando está habilitada la propiedad **Habilitar depuración remota**, corresponde al nombre del equipo en que se ejecutará el archivo .exe. La ubicación del archivo .exe en el equipo remoto debe coincidir con el valor de la propiedad **Ruta de acceso**, que está en la página de propiedades **Generar** de la carpeta **Propiedades de configuración**. El equipo remoto debe tener instalados los componentes del depurador. El usuario del equipo local debe estar incluido en el grupo Usuarios de depuración del equipo remoto. El archivo .exe debe estar en un directorio que se pueda compartir en el equipo remoto.

Para obtener acceso a esta propiedad mediante programación, vea [RemoteDebugMachine](#).

Para obtener información sobre cómo tener acceso a la página de propiedades **Depuración** de la carpeta **Propiedades de configuración**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

### Vea también

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Avanzadas, Propiedades de configuración, Páginas de propiedades de <Nombre de proyecto> (Cuadro de diálogo)

La página de propiedades **Avanzadas** de la carpeta **Propiedades de configuración** contiene las siguientes propiedades:

## Dirección base

Especifica la dirección base preferente en la que se debe cargar un archivo DLL. Vea [/baseaddress](#) para obtener más información.

Vea [BaseAddress \(Propiedad\)](#) para obtener más información sobre cómo obtener acceso mediante programación a esta propiedad.

## Opciones adicionales

Se pueden especificar manualmente opciones del compilador en este cuadro. Si una opción del compilador no está disponible en ninguna página de propiedades ([/ss](#), por ejemplo), se puede especificar aquí. Para obtener más información acerca de las opciones del compilador disponibles, vea [Opciones del compilador de Visual J#](#).

Se recomienda utilizar **Opciones adicionales** para especificar únicamente las opciones del compilador que no están disponibles en ninguna otra parte de la página de propiedades. Esto evitará confusiones si especifica un valor sin darse cuenta en otro lugar de la hoja de propiedades y en **Opciones adicionales**. Un valor especificado en **Opciones adicionales** reemplazará el valor de la misma propiedad definido en otra parte de la página de propiedades. Por ejemplo, si especifica Application1 como la propiedad **AssemblyName** en la carpeta **Propiedades comunes** de la página de propiedades **General** ([General, Propiedades Comunes](#)) y especifica también /out:MyProgram.exe en el cuadro **Opciones adicionales**, el ensamblado resultante creado por el proyecto será MyProgram.exe.

Para obtener información sobre cómo tener acceso a la página de propiedades **Avanzadas** de la carpeta **Propiedades de configuración**, vea [Establecer las propiedades de un proyecto de Visual J#](#).

## Vea también

[Propiedades de un proyecto de Visual J#](#) | [Establecer las propiedades de un proyecto de Visual J#](#)

# Ejemplos de Visual J#

Esta sección contiene descripciones breves de ejemplos de Visual J#. Estas descripciones incluyen un vínculo para abrir o copiar los archivos de código fuente del ejemplo. Cada ejemplo tiene un archivo de solución (.sln); por tanto, se puede abrir, generar y depurar en el entorno de desarrollo de Visual Studio® .NET.

## En esta sección

### [Ejemplos de aplicaciones](#)

Ejemplos que se modelan después de acabar las aplicaciones que pueda desarrollar.

### [Ejemplos basados en varios lenguajes](#)

Ejemplos que utilizan componentes desarrollados en otros lenguajes.

### [Ejemplos generales](#)

Ejemplos adicionales.

### [Ejemplos de tecnología](#)

Ejemplos que muestran tecnologías individuales y características de lenguaje como la invocación de plataforma, la carga de recursos, seguridad, atributos, eventos, delegados y mucho más.

## Secciones relacionadas

Tema	Ubicaciones
<a href="#">Ejemplos</a>	dv_cscon
<a href="#">Ejemplos</a>	dv_cscon
<a href="#">Ejemplos</a>	dv_cscon
<a href="#">Ejemplos</a>	Ejemplos de Visual Basic
<a href="#">Ejemplos</a>	Ejemplos de Visual Basic
<a href="#">Ejemplos</a>	dv_vbcn
<a href="#">Ejemplos</a>	dv_vbcn
<a href="#">Ejemplos</a>	
<a href="#">Ejemplos de Visual C#</a>	Aplicaciones de ejemplo de C#
<a href="#">Ejemplos de Visual C#</a>	Aplicaciones de ejemplo de C#

# Ejemplos de aplicaciones

Los ejemplos de esta sección se modelan después de completar las aplicaciones que se pueden desarrollar. Estos ejemplos muestran un control de errores y un desarrollo orientado a objetos más completos que otros ejemplos de Visual J#.

Ejemplo	Descripción
<a href="#">Acronym Web Service</a>	Muestra la creación y el uso de un servicio Web XML desde una aplicación Web. También muestra cómo enlazar el contenido de un conjunto de datos a una cuadrícula de datos de servidor.
<a href="#">Agent Explorer</a>	Explica cómo actualizar una aplicación de Java o COM para ejecutarla en Visual J#.
<a href="#">Dice</a>	Muestra cómo utilizar clases de C# desde Visual J#.
<a href="#">Photo Album</a>	Aplicación de álbum de fotos de formularios Windows Forms que se puede utilizar para crear, abrir y modificar imágenes JPEG, GIF y BMP.
<a href="#">TypeFinder</a>	Proporciona una interfaz de línea de comandos para obtener información sobre los tipos del entorno de usuario. Asimismo, se proporciona información general sobre la reflexión.
<a href="#">Vjsresgen</a>	Convierte un archivo de recursos de Visual J++ 6.0 en un archivo de recursos de .NET Framework (.resx).
<a href="#">WinTalk</a>	Ofrece una breve introducción a Windows Forms y Sockets, incluidos los temas de FCL. El ejemplo es una aplicación de charla muy sencilla que utiliza sockets.
<a href="#">WordCount</a>	Ofrece una introducción a la biblioteca de clases de .NET Framework Proporciona una aplicación que cuenta las palabras de un archivo de texto y organiza los resultados de acuerdo con argumentos de la línea de comandos.

## Vea también

[Ejemplos de Visual J#](#) | [Ejemplos de tecnología](#) | [Ejemplos generales](#) | [Ejemplos basados en varios lenguajes](#)

# Ejemplo Acronym-WebService (crear y utilizar un servicio Web XML)

Este ejemplo explica cómo:

- Crear un servicio Web XML.
- Utilizar controles Web como botones, etiquetas y cuadrículas de datos.
- Utilice el servicio Web.

El servicio Web toma los primeros caracteres del acrónimo y devuelve todos los acrónimos coincidentes.

La aplicación Web consume este servicio Web y enlaza el contenido del conjunto de datos al control **Datagrid** del servidor.

## Requisitos

- IIS
- Microsoft Visual Studio® .NET
- Microsoft Visual J#

## Generar y ejecutar el ejemplo

### Para preparar la ejecución de este ejemplo

1. Una vez descargados los directorios del ejemplo (AcronymWA y AcronymWebServiceInJavaLanguage) en su equipo, cópielos en %SystemDrive%\inetpub\wwwroot.
2. Convierta AcronymWA en un directorio virtual en IIS siguiendo estos pasos:
  - a. En **Inicio**, elija **Programas, Herramientas administrativas** y, a continuación, **Administrador de servicios Internet**.
  - b. Haga clic con el botón secundario en el sitio Web predeterminado del árbol y elija **Nuevo** y, a continuación, **Directorio virtual**.
  - c. Defina el alias como **AcronymWA**.
  - d. Especifique la ubicación %SystemDrive%\inetpub\wwwroot\AcronymWA.
  - e. Seleccione todos los permisos de acceso (**Lectura, Ejecutar secuencias de comandos, Ejecutar, Escritura y Examinar**).
3. Cree un directorio virtual para AcronymWebServiceInJavaLanguage siguiendo los pasos anteriores.
4. Agregue el permiso **Write** al subdirectorio AcronymWebServiceInJavaLanguage para la cuenta de usuario local ASPNET (<nombre\_equipo>\ASPNET). Esto permite al servicio Web leer y escribir en la base de datos de Microsoft Access Acro.mdb, que se encuentra en este directorio.

**Nota** En el archivo WhatItMeans.aspx.jsl, busque el método **set\_ConnectionString** y reemplace C: por la letra de unidad apropiada, si fuera necesario.

5. Abra Microsoft Visual Studio .NET.
6. Abra AcronymWA.sln desde %SystemDrive%\inetpub\wwwroot\AcronymWA.
7. En el menú **Generar**, haga clic en **Generar**.
8. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* (ratón) en AcronymExplainer.aspx y seleccione **Establecer como página de inicio**.

### Para probar el ejemplo

1. Seleccione **Iniciar sin depurar** en el menú **Depurar**. Verá la página que solicita el acrónimo. A la base de datos Acro.mdb se puede tener acceso desde Microsoft Access con la contraseña **vj#sample**.
2. Escriba algún carácter en el cuadro de edición.
3. Haga clic en el botón **Get Acronym**.
4. Aparecerá una lista de títulos en la **cuadrícula de datos**.

## Vea también

# Ejemplo AgentExplorer (actualizar una aplicación de Java o COM)

En este ejemplo, se muestra cómo se pueden actualizar las aplicaciones de Java o COM de Visual J++<sup>®</sup> 6.0 para ejecutarlas en Visual J#. El ejemplo utiliza contenedores generados por la herramienta JActiveX<sup>®</sup> para el componente COM de control de Microsoft Agent, así como las clases de los paquetes com.ms.\* y JDK nivel 1.1.4. El ejemplo crea una instancia del control COM de Microsoft<sup>®</sup> Agent utilizando el nuevo operador en los contenedores de JActiveX ya generados. A continuación, convierte los objetos COM en interfaces específicas y llama a los métodos y propiedades que contiene. Los contenedores de JActiveX se generan desde el servidor COM agentsvr.dll del directorio %WinDir%\msagent.

## Generar y ejecutar el ejemplo

### Para generar desde el entorno de desarrollo:

1. Escriba **GenerateWrappers.bat** en la línea de comandos.
2. Abra agentexplorer.vjsproj en el entorno de desarrollo y genere el ejemplo (Ctrl+Mayús+B).

### Para generar desde la línea de comandos

- Escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **AgentExplorer** y presione ENTRAR.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de aplicaciones](#)



# Ejemplo Dice (utilizar clases de C#)

En este ejemplo, se muestra cómo utilizar clases de C# desde Visual J#, procediendo como sigue:

- Implementando una interfaz (definida en C#) en una clase de Visual J#.
- Controlando las excepciones iniciadas desde una clase de C# en Visual J#.
- Examinando las propiedades expuestas por una clase de C# en Visual J#.
- Recibiendo los eventos originados por una clase de C# en Visual J#.

El proyecto cscServer implementa un dado (de los que se utilizan en los juegos).

El número de lados (caras) del dado se puede definir mediante programación. El dado se gira con el método **Roll**. La operación **Roll** devuelve el número de la cara superior del dado.

El dado utiliza la interfaz **IRoll** para llevar a cabo la operación Roll. Se necesitan clientes para proporcionar una implementación de esta interfaz. El cliente es necesario para definir esta interfaz en el dado antes de invocar al método **Roll**.

Si se construye el dado con un número de lados incorrecto (por ejemplo, inferior o igual a cero), el dado inicia una excepción **BadDieSizeException**. Si se invoca al método **Roll** en el dado antes de definir el número de lados, el dado inicia una excepción **System.NullReference**.

El dado expone una propiedad de sólo lectura (**NumSides**) que se puede utilizar para inspeccionar el número de lados del dado.

Cada vez que se define una implementación de la interfaz IRoll en el dado, provoca el evento LoadedDice.

El proyecto Dice realiza lo siguiente:

- Proporciona implementaciones para la interfaz **IRoll**.
- Captura las dos excepciones iniciadas.
- Examina la propiedad **NumSides**.
- Proporciona un objeto delegado para recibir los eventos que provoca el dado.

## Generar y ejecutar el ejemplo

### Para generar desde el entorno de desarrollo:

1. Abra el archivo Dice.sln.
2. Genere el ejemplo (Ctrl+Mayús+B).

### Para generar desde la línea de comandos

- Escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **Dice** y presione ENTRAR.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de aplicaciones](#)

# Ejemplo Winforms-Photo-Album (utilizar una biblioteca de formularios Windows Forms)

En este ejemplo se muestra el uso de una biblioteca de formularios Windows Forms utilizando Visual J#. El ejemplo es una aplicación de álbum de fotos que se puede utilizar para crear, abrir y modificar archivos de álbum nuevos y existentes. Abre imágenes de tipo JPEG, GIF y BMP.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

1. Abra winforms\_photo\_album.sln en el entorno de desarrollo.
2. En el menú **Generar**, haga clic en **Generar**.
3. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

### Para crear un álbum nuevo

1. En el menú **File**, haga clic en **New Album**.
2. Haga clic en el botón **Browse** y agregue los archivos que desee.
3. Una vez que se muestren los nombres de archivo en el cuadro de texto adyacente al botón **Browse**, haga clic en el botón **Add** para agregar los archivos al álbum que está creando.
4. Opcional. Modifique el título y la descripción de alguna imagen del siguiente modo:
  - a. Seleccione la imagen.
  - b. Realice los cambios necesarios en el título o en la descripción.
  - c. Haga clic en el botón **Modify**.
5. Opcional. Quite los archivos que no sean necesarios haciendo clic en el botón **Remove**.
6. Una vez que haya terminado de editar, haga clic en el botón **Finish** para crear el álbum.
7. Escriba el nombre del álbum en el cuadro de diálogo que se muestra y guárdelo.

El álbum está listo para verlo.

## Ver el álbum

Puede ver el álbum de tres maneras distintas.

### Presentación automática

- En el menú **View**, haga clic en **Slide Show**.
- Los botones **First** y **Last** le llevarán a la primera y última imagen del álbum.
- Los botones **Previous** y **Next** le llevarán un paso hacia delante o hacia atrás, excepto en la primera y última imagen.
- El cuadro de la parte superior de la imagen muestra el título y el cuadro de la derecha muestra la descripción de la imagen mostrada.

### Vista de miniaturas

- En el menú **View**, haga clic en **Thumbnails**.
- Las imágenes se muestran como miniaturas en el panel emergente. Si hace clic en ellas, verá una imagen más grande a la izquierda. Si hace doble clic, se abrirá una ventana nueva que muestra la imagen con un tamaño más grande.
- Para mostrar sólo el nombre del archivo como una lista sin imágenes, haga clic en **Settings** en el menú **View** y, a continuación, en **List**. Algo similar ocurre si hace clic en **Settings**, en el menú **View**, y después en **Small icons**, ya que no hay iconos pequeños disponibles.

## Manualmente

- En el menú **View**, haga clic en **Auto Slide**.
- Haga clic en el botón **Start** para iniciar la presentación con diapositivas.
- Haga clic en el botón **Pause** para congelar la vista o el botón **Stop** para detener la presentación con diapositivas.
- Puede definir la velocidad a la que se muestran las imágenes con el control **Trackbar**.

## Clases del ejemplo

Las clases siguientes se utilizan este ejemplo.

Archivo	Descripción
Form1.jsl	Ésta es la clase principal. Es una subclase de <b>System.Windows.Forms.Form</b> . Contiene la mayoría de los controles de la interfaz gráfica de usuario necesarios para la aplicación.
CreateAlbum.jsl	Esta clase contiene la interfaz gráfica de usuario y la funcionalidad para crear un álbum nuevo.
PictureBox.jsl	Clase de cuadro de imagen personalizado necesaria para la aplicación.
SliderThread.jsl	Esta clase implementa la funcionalidad de subprocesamiento que controla la velocidad a la que se muestran las imágenes en una presentación automática.
MsgBox.jsl	Clase de utilidad utilizada para mostrar mensajes de error y estado personalizados.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de aplicaciones](#)

# Ejemplo TypeFinder (obtener información de tipos)

Este ejemplo proporciona una interfaz de línea de comandos para obtener información sobre los tipos del entorno. A veces, buscar un espacio de nombres o una DLL de un tipo específico puede resultar difícil y, una vez encontrado el tipo, puede ocurrir que la documentación no esté disponible o no esté actualizada. Esta utilidad proporciona un modo sencillo para determinar los tipos disponibles, el módulo en el que se encuentran y las interfaces, métodos, campos, propiedades y eventos disponibles en el tipo. Este ejemplo proporciona también una introducción a la reflexión.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar el ejemplo desde el entorno de desarrollo

- 1. Abra el archivo TypeFinder.sln.
- 2. Genere el ejemplo (Ctrl+Mayús+B).
- 3. Presione F5 para ejecutar el ejemplo.

### Para generar y ejecutar el ejemplo desde la línea de comandos:

- 1. Escriba **BUILD.bat**.
- 2. Escriba **TypeFinder** y presione ENTRAR.
- 3. Para buscar la ubicación de cada tipo con una palabra del nombre, escriba **TypeFinder palabra** desde la línea de comandos.

## Clases y tecnologías del ejemplo

Las clases y tecnologías siguientes utilizan este ejemplo.

### Reflexión

Clase o tecnología	Descripción
<b>System.Reflection.Assembly</b>	Carga ensamblados en el dominio de aplicación para poder buscar tipos e n ellos.
<b>System.Reflection.Module</b>	Obtiene tipos del ensamblado o módulo por comparación con la cadena d e búsqueda.
<b>System.Type</b>	Obtiene información de tipo como el nombre, el espacio de nombres y los miembros.
<b>System.Reflection.PropertyInfo</b>	Busca información sobre las propiedades en los tipos.
<b>System.Reflection.EventInfo</b>	Busca información sobre los eventos en los tipos.
<b>System.Reflection.FieldInfo</b>	Busca información sobre los campos en los tipos.
<b>System.Reflection.MethodInfo</b>	Busca información sobre los métodos de los tipos.

### E/S

Clase o tecnología	Descripción
<b>System.IO.TextWriter</b>	Utilizada por el tipo de ejemplo <code>IndentedWriter</code> para mostrar el resultado (en la consola de manera predeterminada) de un modo genérico.

### Texto

Clase o tecnología	Descripción
<b>System.Text.StringBuilder</b>	Utilizada por el tipo de ejemplo <code>IndentedWriter</code> para crear una cadena.
<b>System.String</b>	Utilizada en el ejemplo para buscar cadenas de formato, subcadenas, cade nas en mayúscula, etc.

### Colecciones

Clase o tecnología	Descripción
<b>System.Collections.ArrayList</b>	Utilizada en el ejemplo para administrar una lista de cadenas .

### Registro

Clase o tecnología	Descripción
--------------------	-------------

<b>Microsoft.Win32.Registry</b>	Utilizada para crear una instancia del tipo <code>RegistryKey</code> para una subclase de la clave <code>LocalMachine</code> .
<b>Microsoft.Win32.RegistryKey</b>	Utilizada para leer valores de una clave del Registro.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar `vsvars.bat` (que se encuentra en el directorio `<%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools`). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio *<raíz de instalación de Microsoft Visual J++ 6.0>*.

## Vea también

[Ejemplos de aplicaciones](#)

# Ejemplo Vjsresgen (convertir un archivo de recursos de Visual J++ 6.0 en un archivo de recursos de .NET Framework (.resx))

Este ejemplo muestra cómo convertir un archivo de recursos de Visual J++ 6.0 en un archivo de recursos de .NET Framework (.resx).

Para que la aplicación actualizada funcione correctamente, es importante que todos los recursos de CLASSPATH que utiliza la aplicación estén identificados e incrustados o enlazados en el ensamblado. Se debe tener cuidado para mantener la estructura de directorios al convertir los archivos de recursos en archivos de recursos de .NET Framework.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra vjsresgen.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

1. Este ejemplo se debe ejecutar desde la línea de comandos. Escriba **vjsresgen.exe** en la línea de comandos con los nombres de los archivos que se van a convertir al formato .resx.
2. De manera predeterminada, el archivo .resx generado se denomina vjs.resx. Puede reemplazar este nombre utilizando la opción **/out**.

## Sintaxis y opciones de Vjsresgen

```
vjsresgen [/out:filename] [/recurse:wildcard] resourcefiles
```

donde:

*filename*

Nombre del archivo de resultados .resx que se va a crear.

*wildcard*

Expresión comodín que debe coincidir con el nombre de archivo.

*resourcefiles*

Archivos de recursos de entrada.

La opción **/recurse** permite especificar archivos que coincidan con una expresión comodín (*wildcard*). De manera opcional, puede especificar el directorio en el que desea que comience la búsqueda. Si no se especifica, la búsqueda empieza en el directorio del proyecto.

## Ejemplos de ejecución de Vjsregen.exe

Los ejemplos siguientes muestran cómo utilizar Vjsresgen.exe desde la línea de comandos.

### Recursos en el mismo directorio

1. Vjsresgen /out:myresource.resx \*.txt
  2. Resgen myresource.resx myresource.resources
  3. Jblmp /res:myresource.resources myarchive.zip
- O bien
- vjc /resource:myresource.resources \*.java

### Recursos en subdirectorios

1. Vjsresgen /out:myresource.resx subdir\res1.bmp subdir\r.resources subdir2\res.txt subdir2\r.resources
2. Resgen myresource.resx myresource.resources
3. Jblmp /res:myresource.resources myarchive.zip

O bien

```
vjc /resource:myresource.resources *.java
```

En el ejemplo anterior, `subdir\r.resources` y `subdir2\r.resources` son dos archivos de recursos diferentes. Cuando se llama a **Class.getResource** en una clase que existe en el paquete `subdir`, se obtiene el recurso especificado por `subdir\r.resources`, mientras que una clase del paquete `subdir2` obtiene `subdir2\r.resources`.

En el ejemplo anterior, `r.resources` es un archivo creado en Visual J++ 6.0 y no es equivalente al formato de archivo `.resources` compatible con .NET Framework.

## Recursos que utilizan la opción /recurse

1. `Vjsresgen /out:my.resx /recurse:p1\*.resources /recurse:p2\*.bmp`
2. `Resgen my.resx`
3. `Jblmp /res:my.resources /recurse`

O bien

```
vjc /resource:my.resources /recurse:*.java
```

En el ejemplo anterior, los archivos `*.resources` presentes en el directorio `p1` y los archivos `*.bmp` presentes en el directorio `p2` están incrustados.

**Nota** Puesto que la ruta del recurso convertido que se incrustó en el archivo de recursos administrados (`.resx`) debe coincidir con la ruta utilizada para especificar el recurso en **Class.getResource** (o métodos similares), es posible que deba modificar el código de la aplicación en consecuencia. Por ejemplo, para convertir un recurso de Visual J++ ubicado en `c:\resources\package1\data.txt`, es posible que deba buscar el directorio `c:\temp` y ejecutar después `Vjsresgen`. A continuación, debe modificar el código de la aplicación para definir el recurso como `"package1.data.txt"` en métodos como **Class.getResources(String nombreRecurso)**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar `vsvars.bat` (que se encuentra en el directorio `<%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools`). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio `<raíz de instalación de Microsoft Visual J++ 6.0>`.

## Vea también

[Ejemplos de aplicaciones](#) | [Actualizar aplicaciones de Visual J++ 6.0 que utilizan recursos](#)

# Ejemplo WinTalk (incluye formularios Windows Forms y sockets)

Este ejemplo ofrece una breve introducción a los formularios Windows Forms y sockets, incluidos los temas de la biblioteca de clases de Framework (FCL). El ejemplo es una aplicación de charla muy sencilla que utiliza sockets.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- Escriba **build.bat** desde la línea de comandos.

### Para ejecutar este ejemplo

- Escriba **WinTalk.exe /?** en la línea de comandos para obtener una lista completa de las opciones de la línea de comandos para la utilidad.
- Para probar el ejemplo donde el cliente y el servidor están en el mismo equipo, sólo tiene que ejecutar la aplicación dos veces sin parámetros.
- Cuando especifique un número de puerto, compruebe que otro programa no esté usando ya el puerto; de lo contrario, no se ejecutará correctamente. Pruebe utilizando un número de puerto alto entre 5001 y 5150.
- Para que el ejemplo funcione bien, debe definir correctamente las variables de entorno **Path**, **Include** y **Lib**. Puede definir estas variables ejecutando el archivo CorVars.bat, que podrá encontrar en el directorio `<raízDe\SDK>\Bin`. CorVars.bat se debe ejecutar en todos los shell de comandos.

## Clases y tecnologías del ejemplo

Las clases y tecnologías siguientes se utilizan este ejemplo.

### Windows Forms

Clase o tecnología	Descripción
<b>Application</b>	Se utiliza en la aplicación de ejemplo para implementar un bombeo de mensajes.
<b>Splitter</b>	Tipo de control que supervisa la semántica de divisor. Se utiliza en la aplicación para separar los marcos de texto de envío y recepción.
<b>Panel</b>	Permite la organización, acople y delimitación flexibles de los controles del formulario. En este caso, el panel contiene los dos cuadros de texto y el divisor. El control de estado, por otro lado, es un par del panel.
<b>TextBox</b>	Implementa dos controles de edición en el ejemplo. Uno es de sólo lectura para recibir texto.
<b>Label</b>	Se utiliza para mostrar información de estado estática en el ejemplo.
<b>MessageBox</b>	Se utiliza en el ejemplo para transmitir información sobre errores y situaciones de cierre.

### Funciones de red

Clase o tecnología	Descripción
<b>EndPoint</b>	Se utiliza en el ejemplo para encapsular una dirección IP y un número de puerto.
<b>IPAddress</b>	Proporciona lógica relacionada con <b>IPAddress</b> muy útil.
<b>Dns</b>	Se utiliza para obtener una dirección IP desde un nombre DNS.
<b>Socket</b>	La mayor parte de la funcionalidad de red implementada en código administrado se realiza con una abstracción como servicios Web XML. Sin embargo, es posible implementar código de socket directamente y este ejemplo utiliza la clase <b>Socket</b> para ello.
<b>NetworkStream</b>	Se deriva de <b>stream</b> y utiliza un socket como dispositivo subyacente. Esto lo hace muy útil para secuenciar datos a través de un socket. Puede utilizar el tipo <b>NetworkStream</b> igual que haría con cualquier otro tipo de secuencia, como <b>FileStream</b> .

### E/S

Clase o tecnología	Descripción
<b>StreamWriter</b>	Se utiliza para escribir en la secuencia de red que representa un socket.
<b>StreamReader</b>	Se utiliza leer de la secuencia de red que representa un socket.



## Recolección de elementos no utilizados

Clase o tecnología	Descripción
<b>GC</b>	Se utiliza para suprimir la finalización de un objeto personalizado que ya se ha eliminado.

## Delegados

Clase o tecnología	Descripción
<b>MulticastDelegate</b>	Este ejemplo implementa un tipo de delegado derivado de <b>MulticastDelegate</b> . Este tipo se utiliza para notificaciones relacionadas con eventos de red. El ejemplo utiliza este tipo de delegado para enlazar el tipo de red con el tipo de formulario del ejemplo.

## Subprocesamiento

Clase o tecnología	Descripción
<b>ThreadPool</b>	La clase personalizada <i>Talker</i> de este ejemplo controla la lógica de red y utiliza un segundo subproceso para leer de la red. Sin embargo, en lugar de crearlo, utiliza un subproceso de Common Language Runtime. Éste es el método recomendado para el subprocesamiento múltiple en código administrado.

## Excepciones

Clase o tecnología	Descripción
<b>IOException</b>	Los tipos <b>StreamReader</b> y <b>StreamWriter</b> suelen iniciar el tipo <b>IOException</b> . Sin embargo, para capturar y controlar estas excepciones correctamente, suele ser necesario comprobar la excepción asociada que refleja el error de base. En el caso de este ejemplo, éste será el tipo <b>SocketException</b> .
<b>SocketException</b>	Este ejemplo captura el tipo <b>SocketException</b> y comprueba la propiedad <b>ErrorCode</b> del socket para determinados tipos de error esperados.

## Requisitos de la línea de comandos

Para que funcionen adecuadamente los ejemplos de Visual J# al generarlos y ejecutarlos desde la línea de comandos, debe definir correctamente la variable de entorno **Path**. Debe definir la variable **Path** para incluir el directorio `<%windir%>\Microsoft .NET\Framework\v<%versión%>`. También es posible que deba ejecutar `CorVars.bat` (ubicado en el directorio `<RaízDe.NETFrameworkSDK>\Bin`).

## Vea también

[Ejemplos de aplicaciones](#)

# Ejemplo WordCount (introducción a la biblioteca de clases de .NET Framework)

Este ejemplo ofrece una introducción a la biblioteca de clases de .NET Framework. Muestra cómo crear una aplicación que abra varios archivos (especificados en la línea de comandos) y cuenta los bytes, caracteres, palabras y líneas de cada archivo. Se muestran los resultados de cada archivo y el total de todos los archivos.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar el ejemplo desde el entorno de desarrollo

1. Abra el archivo WordCount.sln.
2. Genere el ejemplo (Ctrl+Mayús+B).
3. Presione F5 para ejecutar el ejemplo.

### Para generar y ejecutar el ejemplo desde la línea de comandos

1. Escriba **BUILD.bat**.
2. Escriba **WordCount** y presione ENTRAR.
3. Para contar las palabras de un archivo, escriba **WordCount nombreArchivo**.

## Clases y tecnologías del ejemplo

Las clases y tecnologías siguientes se utilizan este ejemplo.

### E/S

Clase o tecnología	Descripción
<b>System.IO.FileStream</b>	Se utiliza para el acceso a archivos. Esta clase se utiliza para leer y escribir archivos.
<b>System.IO.StreamWriter</b>	Se utiliza cuando los argumentos de la línea de comandos requieren que el resultado se envíe a un archivo. <b>StreamWriter</b> se utiliza junto con una instancia de <b>FileStream</b> para dar formato al texto del archivo de resultados.
<b>System.IO.StreamReader</b>	Se utiliza para leer el contenido de un archivo para el que el ejemplo está contando las palabras. <b>StreamReader</b> se utiliza junto con una instancia de <b>FileStream</b> para leer el texto de un archivo.

### Colecciones

Clase o tecnología	Descripción
<b>System.Collections.ArrayList</b>	Se utiliza como clase de colección general para almacenar conjuntos de objetos.
<b>System.Collections.SortedList</b>	Se utiliza como clase de colección para almacenar conjuntos de objetos de un modo ordenado.
<b>System.Collections.IEnumerator</b>	Se utiliza directamente para enumerar conjuntos de objetos.
<b>System.Collections.IDictionaryEnumerator</b>	Se utiliza directamente para enumerar conjuntos de objetos indizados.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

### Vea también



# Ejemplos basados en varios lenguajes

Los ejemplos de esta sección proceden de ejemplos de otros lenguajes.

Ejemplo	Descripción
<a href="#">DataSetConsumer</a>	Muestra cómo utilizar conjuntos de datos de varias tablas en un cliente SOAP generado por SPROXY. El código de Visual J# crea un conjunto de datos de varias tablas y lo expone mediante un método de servicios Web XML.
<a href="#">Delegates</a>	Muestra el uso de delegados en el componente Extensiones administradas de C++. Se crea un delegado con un método definido en una DLL creada con Visual J#.
<a href="#">JrnlPost</a>	Crea un único archivo DLL que contiene lógica empresarial nativa, una interfaz y una biblioteca de tipos para clientes COM y un ensamblado con los metadatos correspondientes para clientes .NET Framework. El código de Visual J# crea una instancia de un objeto administrado del archivo DLL que llama a un par de métodos. Ilustra que el archivo DLL contiene tanto clases COM como clases .NET Framework.
<a href="#">ManagedEvents</a>	Muestra la interoperabilidad de control de eventos entre las Extensiones administradas de C++ y los objetos de Visual J#. El código de Visual J# implementa tanto el origen como el receptor de un evento.
<a href="#">MCppWrapper</a>	Ilustra cómo una clase <b>__gc</b> de extensiones administradas puede utilizarse como una clase de proxy (o contenedor) para una clase C++ no administrada en un archivo DLL. El código de Visual J# es un cliente que lee una cadena y una posición desde la consola y muestra un sufijo.
<a href="#">OnlineAddressBook</a>	Almacena una libreta de direcciones en un servicio Web XML creado con el servidor ATL. En el ejemplo se muestra un cliente de Visual J# que llama al servicio Web XML.
<a href="#">TilePuzzle</a>	Muestra la interoperabilidad entre los componentes administrados (escritos con las Extensiones administradas de C++ y Visual J#) y los componentes nativos (escritos con C++ utilizando atributos COM).

## Vea también

[Ejemplos de Visual J#](#) | [Ejemplos de aplicaciones](#) | [Ejemplos de tecnología](#) | [Ejemplos generales](#)

# Ejemplo DataSetConsumer (utiliza conjuntos de datos de varias tablas)

Este ejemplo muestra cómo utilizar varios conjuntos de datos de tablas desde un cliente de Visual C++ no administrado mediante un proxy de servicio Web XML.

El ejemplo DataSetConsumer ilustra cómo utilizar varios conjuntos de datos (DataSets) de tablas desde una aplicación de Visual C++ no administrada. DataSetConsumer es una aplicación de servidor ATL que utiliza un conjunto de datos de tabla múltiple expuesto a través de un servicio Web XML creado con ASP.NET.

DataSetConsumer muestra también cómo utilizar referencias Web para atender al uso de varios conjuntos de datos de tablas. Las referencias Web convierten los conjuntos de datos de tablas simples en estructuras de C++, pero tratan los conjuntos múltiples como flujos XML. Por ello, el método de servicios Web XML correspondiente en la clase de proxy devuelve la serialización XML completa del conjunto de datos en forma de cadena. A continuación, debe ser analizada y descompuesta para que sea útil en la mayoría de las aplicaciones.

DataSetConsumer se corresponde directamente con el tema

[Tutorial: Aplicaciones de Visual C++ que utilizan datos de componentes de .NET Framework](#). La aplicación DataSetConsumer tiene acceso a varias tablas de un conjunto de datos, mientras que el tutorial sólo puede obtener acceso a una única tabla.

## Requisitos

- IIS debe estar instalado y ejecutándose.
- Debe haber instalada al menos una base de datos SQL Server. Si no dispone de las herramientas de cliente de SQL Server 2000, use la herramienta descrita en [Herramienta de instalación de base de datos de ejemplo](#).

## Generar y ejecutar el ejemplo

Para utilizar este ejemplo, siga los pasos de esta sección. Creará un proyecto de servicio Web XML denominado WebService1 que define un método de servicio Web para el acceso a los datos desde un conjunto de datos. A continuación agregará una referencia Web que creará una clase de proxy para el servicio Web XML. DataSetConsumer utiliza una instancia de este proxy para llamar al método de servicios Web XML en WebService1. WebService1 devuelve el conjunto de datos como tipo **BSTR** y DataSetConsumer lo carga como XML, utilizando DOM y MSXML para analizar la cadena e imprimir los datos como HTML.

### Para crear un proyecto de servicio Web ASP.NET para el acceso a un conjunto de datos

1. Cree un nuevo servicio Web XML con ASP.NET.

En el menú **Archivo**, haga clic en **Nuevo** y, a continuación, en **Proyecto**. Aparecerá el cuadro de diálogo **Nuevo proyecto**. En la carpeta **Proyectos de Visual J#**, seleccione **Servicio Web ASP.NET**. Dé al proyecto el nombre `WebService1`. La ubicación debe ser la predeterminada (`http://localhost/WebService1`).

Haga clic en **Aceptar**. Cuando haya creado el proyecto, verá `Service1.asmx.jsl` en el panel de diseño.

2. Cree una conexión de datos.

En el Explorador de servidores, haga clic con el botón secundario en **Conexión de datos** y, a continuación, en **Agregar conexión**. Aparecerá el cuadro de diálogo **Propiedades de vínculo de datos**. Rellene los datos para crear una conexión de datos con una base de datos, por ejemplo, Northwind. Active la casilla de verificación **Permitir guardar contraseña**.

3. Agregue tablas y adaptadores de datos.

Abra el nodo **Tablas** de la nueva conexión de datos y arrastre dos tablas del Explorador de servidores al panel de diseño. Aparecerán dos adaptadores de datos y una conexión de datos en el panel de diseño.

4. Genere un conjunto de datos.

En el menú **Datos**, haga clic en **Generar conjunto de datos**. En el cuadro de diálogo **Generar conjunto de datos**, elija **Nuevo conjunto de datos**, dé al conjunto el nombre predeterminado (`DataSet1`), y elija agregar ambas tablas al conjunto de datos. Active también la casilla **Agregar este conjunto de datos al diseñador** (opcional, pero recomendado). Haga clic en **Aceptar** para crear el conjunto de datos (denominado `dataSet11` de forma predeterminada).

5. Implemente un método de servicios Web.

En el menú **Ver**, haga clic en **Código**. Aparecerá el código de `Service1.asmx.jsl` en el panel de edición. Sitúese al final del

archivo, en el ejemplo de método de servicios Web HelloWorld:

```
// /** @attribute WebMethod() */  
// public System.String HelloWorld()  
// {  
//     return "Hello World";  
// }
```

En lugar del código del ejemplo HelloWorld, implemente un nuevo método de servicios Web denominado GetDataSet de este modo:

```
/** @attribute WebMethod() */  
public DataSet1 GetDataSet()  
{  
    this.sqlDataAdapter1.Fill (this.dataSet11);  
    return dataSet11;  
}
```

**Nota** En este código se presupone que utiliza una base de datos de SQL Server.

6. Genere el proyecto WebService1. En el menú **Generar**, haga clic en **Generar solución**.

#### Para utilizar datos de DataSetConsumer

1. Abra el proyecto DataSetConsumer.
2. Abra el archivo de solución, DataSetConsumer.sln, en el entorno de desarrollo de Visual Studio.
3. Asegúrese de que DataSetConsumer.h contiene esta línea:

```
#include "WebService.h"
```

4. Agregue una referencia Web al proyecto.

Con ello se creará una clase de proxy para el servicio Web XML. Un cliente de servicio Web XML puede llamar a un método del proxy que, a su vez, hace lo necesario para invocar de forma remota a través de la red al método correspondiente de servicios Web XML. De manera predeterminada, la clase del proxy utiliza SOAP para tener acceso al método correspondiente. SOAP admite el completo conjunto de tipos de datos de los tres protocolos compatibles (SOAP, HTTP-GET y HTTP-POST). Agregar una referencia Web puede generar también clases de proxy para los dos últimos protocolos. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* (ratón) en el nodo de proyecto DataSetConsumer y seleccione **Agregar referencia Web** en el menú contextual.

Aparecerá el cuadro de diálogo **Agregar referencia Web**.

5. En la barra de dirección, escriba la dirección URL de WebService1 y presione ENTRAR:

```
http://localhost/WebService1/Service1.asmx
```

La descripción de servicio Web es un documento XML escrito con una gramática de XML denominada WSDL (*Web Service Description Language*, lenguaje de descripción de servicio Web) que define el formato de los mensajes para el servicio Web XML.

6. Haga clic en **Agregar referencia**. De esta forma se agregará la clase de proxy Service1 al proyecto DataSetConsumer. El archivo de encabezado WebService.h aparecerá en el **Explorador de soluciones**.

**Nota** En WebService.h, el constructor CService1T incluye directamente en su código la dirección URL especificada al agregar la referencia Web. Si desea utilizar DataSetConsumer con otro servicio Web XML, cambie la dirección URL de modo que concuerde con la ubicación del archivo .asmx de ese servicio.

7. En el menú **Generar**, haga clic en **Generar solución** para generar el proyecto DataSetConsumer.
8. Para ejecutar la aplicación cliente, en el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Se le pedirá una dirección URL.

9. Escriba lo siguiente:

```
http://localhost/DataSetConsumer/DataSetConsumer.srf
```

10. Haga clic en **Aceptar**.

Aparecerá el explorador Web con los datos a los que se obtiene acceso a través del servicio Web XML.

## Clases, tecnologías y conceptos del ejemplo

- [Agregar y quitar referencias Web](#)
- [Conjuntos de datos ADO.NET](#)
- [Crear clientes de servicios Web XML](#)
- [DataSet \(Clase\)](#)
- [request\\_handler](#)
- [soap\\_handler](#)
- [soap\\_method](#)
- [Guía de usuario de XML DOM](#)

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplo Delegates (delegados personalizados)

Este ejemplo implementa dos usos de un delegado (invocar un solo método e invocar varios métodos) mediante la palabra clave `__delegate`. Para obtener más información sobre los delegados, vea Delegados de la Extensiones administradas de C++.

En el ejemplo también se ilustra la interoperabilidad entre una aplicación basada en las Extensiones administradas de C++ y un archivo DLL de Visual J# (`j_sdel`). La DLL de Visual J# implementa una clase básica de Visual J# (`j_sdel`) con un método personalizado.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo utilizando Visual Studio

1. En el IDE de Visual Studio, cargue el archivo de solución Delegates.sln.
2. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* en la solución **Delegates**.
3. En el menú contextual, haga clic en **Generar solución**.
4. En el menú **Depurar**, haga clic en **Iniciar**.

Cuando se genera y ejecuta el ejemplo, se crean e invocan delegados que invocan un sólo método o varios métodos:

- `pSCDelegate`

Se crea un delegado para un sólo método y se invoca al método enlazado. La confirmación es una cadena impresa con el valor pasado por el delegado para un sólo método.

- `pMCDelegate`

Antes de crear un delegado para varios métodos, se crea una instancia de la clase de Visual J# (`j_sdel`). A continuación, se enlaza el delegado a los métodos de Visual J# y de clase administrada, y se invoca. La confirmación es una cadena impresa con el valor pasado por el delegado para varios métodos.

## Vea también

[Ejemplos basados en varios lenguajes](#)



# Ejemplo JrnlPost (tener acceso a lógica empresarial desde clientes COM y .NET Framework)

Este ejemplo crea un solo archivo DLL que contiene lógica comercial nativa, una interfaz y una biblioteca de tipo para clientes COM, y un ensamblado con los metadatos correspondientes para clientes de .NET Framework. Los clientes escritos con Visual J# y las Extensiones administradas de C++ se utilizan para tener acceso a la lógica empresarial.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

1. En el IDE de Visual Studio, cargue el archivo de solución JrnlPost.sln.
2. En el Explorador de soluciones, haga clic con el botón secundario del *mouse* en la solución **JrnlPost**.
3. En el menú contextual, haga clic en **Generar solución**.

### Para ejecutar este ejemplo

1. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* en el proyecto comJrnlClient.
2. En el menú contextual, haga clic en **Depurar** y, a continuación, haga clic en **Iniciar nueva instancia**.
3. Repita los pasos 1 y 2 con los proyectos netJrnlClientA y netJrnlClientB.

Cuando se inician los clientes, aparecen una serie de cuadros de diálogo (con un botón Aceptar) que le notifican las llamadas de la lógica empresarial. Para cerrarlos, haga clic en el botón Aceptar.

**Nota** Para obtener más información sobre la interacción entre componentes, recorra paso a paso el código estableciendo puntos de interrupción o utilizando el menú **Debug** para analizar detenidamente el código de ejemplo.

## Descripción del proyecto

El objetivo principal del ejemplo consiste en el acceso a la lógica empresarial desde clientes COM y clientes de .NET Framework, no la implementación de la lógica empresarial en sí. Por tanto, no nos centraremos en la lógica empresarial de este ejemplo. De hecho, la lógica empresarial realiza una validación mínima y muestra cuadros de mensaje de Win32 correspondientes a acciones empresariales de servidor para indicar los pasos. La clase `JEPPost`, que implementa la lógica empresarial está definida e implementada en los archivos `JEPPost.h` y `JEPPost.cpp` respectivamente.

La clase que implementa el componente COM contiene un puntero a una instancia de la clase de la lógica empresarial. Se puede considerar que el componente COM es como el cableado por el que un cliente COM tiene acceso a la lógica empresarial subyacente. `JrnlPost.dll` contiene una biblioteca de tipos que los clientes COM utilizan para descubrir las interfaces y los métodos expuestos por el componente COM. El cliente COM de este ejemplo utiliza la característica **#import** del compilador para utilizar una biblioteca de tipos y generar un puntero inteligente que después utiliza el cliente para crear una instancia y llamar al componente COM. La clase que implementa el componente COM es `comJEPPost`, que está definida e implementada en los archivos `comJEPPost.h` y `comJEPPost.cpp` respectivamente.

La clase que implementa el componente de .NET Framework también contiene un puntero a una instancia de la clase de la lógica empresarial. De manera similar a su homólogo COM, se puede considerar al componente de .NET Framework de este ejemplo como el cableado por el que un cliente .NET Framework tiene acceso a la lógica empresarial subyacente. `JrnlPost.dll` contiene un solo archivo de ensamblado con los metadatos correspondientes que utilizan los clientes de .NET Framework para descubrir y tener acceso a las clases y los métodos públicos expuestos por el componente de .NET Framework. La clase que implementa el componente .NET Framework es `netJEPPost`, que está definida e implementada en los archivos `comJEPPost.h` y `comJEPPost.cpp` respectivamente.

Este ejemplo contiene tres proyectos cliente:

- `ComJrnlClient`, un cliente COM implementado en C++ nativo.
- `NetJrnlClientA`, un cliente de .NET Framework programado en Visual J#.
- `NetJrnlClientB`, un cliente programado con las Extensiones administradas de C++.

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplo ManagedEvents (crear y utilizar eventos en una aplicación administrada)

Este ejemplo muestra la interoperabilidad de control de eventos entre las Extensiones administradas de C++ y los objetos de Visual J#.

## Generar y ejecutar el ejemplo

1. Abra el archivo de solución **ManagedEvents.sln**.
2. Haga clic con el botón secundario del *mouse* (ratón) en el nodo **Receiver** en el **Explorador de soluciones** y seleccione **Establecer como proyecto de inicio**.
3. En el menú **Generar**, haga clic en **Generar solución**.
4. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.
5. También se puede ejecutar el ejemplo con JSEvent como proyecto de inicio. Defina JSEvent como proyecto de inicio.
6. Haga clic con el botón secundario en el proyecto JSEvent y seleccione **Propiedades**. En la ficha **General**, elija el tipo de resultado **Aplicación de consola**.
7. Genere y ejecute el ejemplo.

## Cómo funciona el ejemplo

El código crea dos orígenes de eventos: un origen de eventos de las Extensiones administradas de C++ `Event` (vea `Event.cpp`) y un origen de eventos de Visual J# `JSEvent` (vea `JSEvent.jsl`). También crea dos receptores de eventos: un receptor de eventos de las Extensiones administradas de C++ `Receiver` (vea `Receiver.cpp`) y un receptor de eventos de las Extensiones administradas de C++ `JR2` (vea `JSEvent.jsl`).

El origen de eventos `Event` de las Extensiones administradas de C++ declara dos eventos: `MyEvent` y `MyEvent2`. El origen de eventos `JEvent` de Visual J# declara un evento: `JMyEvent`.

El receptor de eventos `Receiver` de las extensiones administradas de C++ enlaza:

- `Event::MyEvent` a los métodos de controlador de eventos `Handler1` y `Handler2`
- `Event::MyEvent2` con el método de controlador de eventos `Handler5`
- `JEvent::JMyEvent` con los métodos de controlador de eventos `Handler3` y `Handler4`

El receptor de eventos `JR2` de Visual J# enlaza:

- `JEvent::JMyEvent` con los métodos de controlador de eventos `H1` y `H2`
- `Event::MyEvent` a los métodos de controlador de eventos `H3` y `H4`

El código principal de `Receiver.cpp` crea una instancia de los orígenes de eventos y del receptor de eventos de Extensiones administradas de C++. A continuación, llama a los métodos para enlazar los controladores de eventos con los orígenes de eventos y para iniciar estos eventos.

El código principal de `JSEvent.jsl` crea una instancia de los orígenes de eventos y del receptor de eventos de Visual J#. A continuación, llama a los métodos para enlazar, iniciar y desenlazar los controladores de eventos.

Los receptores de eventos `JR2` de Visual J# y el receptor de eventos `Receiver` de las Extensiones administradas de C++ reciben eventos de los orígenes de eventos tanto de las Extensiones administradas de C++ como de Visual J#, `Event` y `JEvent`, respectivamente.

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplo MCppWrapper (contener una DLL de C++)

Este ejemplo ilustra cómo se puede utilizar la clase `__gc` de las Extensiones administradas como clase de proxy (o contenedora) para una clase C++ no administrada de un archivo DLL. Algunos de los conceptos básicos ilustrados son la interoperabilidad de las Extensiones administradas, las clases de recubrimiento y clases proxy, y el cálculo de referencias básico.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo utilizando Visual Studio

1. En el IDE de Visual Studio, cargue el archivo de solución MCppWrapper.sln.
2. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* (ratón) en la solución **MCppWrapper**.
3. En el menú contextual, haga clic en **Generar solución**.
4. En el menú **Depurar**, haga clic en **Iniciar**.

## Cómo funciona el ejemplo

El archivo DLL no administrado proporciona una clase denominada `substring`. Esta clase tiene un método denominado `suffix` que devuelve el sufijo de una cadena a partir de una posición de índice al final de la cadena. El primer carácter de la cadena tiene la posición de índice 1. Una posición de 0 o una posición negativa da como resultado que se devuelva la cadena entera. Una posición superior a la longitud de la cadena hace que se devuelva una cadena vacía.

La clase de subcadena se "envuelve" mediante una clase `__gc` de Extensiones administradas denominada `substring_w`. Esta clase contiene un método denominado `find_suffix` que utiliza una cadena administrada y un valor de tipo **int** como argumentos. El método realiza cálculo de referencias sencillo: la cadena administrada se convierte en una cadena no administrada. Crea un objeto de la clase `substring` y llama a `suffix` con la cadena convertida y una posición de tipo integer. La cadena devuelta se convierte implícitamente en una [String](#) administrada.

El cliente de la clase `substring_w` es un programa sencillo escrito en Visual J# que lee una cadena y su posición en la consola y, después, crea un objeto `substring_w`, llama a `find_suffix` y muestra el sufijo.

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplo OnlineAddressBook (almacena una libreta de direcciones en un servicio Web XML)

Este ejemplo muestra el uso del estado de la sesión para almacenar una libreta de direcciones en un servicio Web XML creado con el servidor ATL.

El ejemplo OnlineAddressBook utiliza clases de plantillas de cliente OLE DB para almacenar una libreta de direcciones de usuario y permitir al usuario definir y recuperar la información mediante un servicio Web XML creado con el servidor ATL.

## Requisitos

Este ejemplo requiere IIS y Microsoft Office XP.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

1. Abra el archivo de solución, OnlineAddressBook.sln, en el entorno de desarrollo de Visual Studio.
2. Modifique el archivo OnlineAddressBookWS.disco y el nombre de equipo de destino de la implementación del proyecto para especificar el equipo que desee.  
  
O bien  
  
Ejecute todos los proyectos sin modificaciones implementando y ejecutando todo en localhost.
3. Copie la base de datos de Microsoft Access incluida, OnlineAddressBookWS.mdb, a la unidad C:\.  
  
O bien  
  
Puede copiarla en una ubicación distinta, siempre que modifique `MYDATASOURCE`, que se define en el archivo OnlineAddressBookWS\OnlineAddressBookWS.h.
4. Genere **OnlineAddressBookWSIsapi** y **OnlineAddressBookWS**.  
  
Se crea un directorio virtual en el equipo en el que se implementan los archivos relevantes.
5. Genere el proyecto de Visual J# incluido, OnlineAddressBookJSharpClient. Puede utilizar este cliente para importar su libreta de direcciones personal desde Microsoft Outlook XP y cargarla en el servidor (o en cualquier servidor que ejecute este servicio Web XML). También puede crear sus propios contactos, cargarlos en el servicio Web XML y descargarlos más tarde desde cualquier sitio.
6. Genere OnlineAddressBookSRF. Con ello se implementará un proyecto de servidor ATL con un cliente de servicio Web XML integrado. Una vez creada una cuenta y agregados algunos registros en la libreta de direcciones mediante el cliente de Visual J#, puede utilizar este proyecto para ver los contactos con un explorador. Esta página debería estar en `http://localhost/OnlineAddressBookWS/OnlineAddressBookSRF.srf`.

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplo TilePuzzle: interoperabilidad entre Visual J# y las Extensiones administradas de C++

Este ejemplo muestra la interoperabilidad entre los componentes administrados (escritos con las Extensiones administradas de C++ y Visual J#) y los componentes nativos (escritos con C++ utilizando atributos COM).

El ejemplo implementa un puzzle básico denominado Tile Puzzle. Carga un mapa de bits, lo divide en cierto número piezas (determinado por el usuario) y las desordena. A continuación, el usuario resuelve el puzzle desplazando cada pieza hasta que aparece la imagen original. Además de estas características, el ejemplo tiene la capacidad de resolver el puzzle utilizando algoritmos de búsqueda heurística programados con las Extensiones administradas de C++ y las clases de .NET Framework.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar TilePuzzle en Visual Studio

1. En el IDE de Visual Studio, cargue el archivo de solución Puzzle.sln.
2. En el **Explorador de soluciones**, haga clic con el botón secundario del *mouse* en la solución Puzzle.
3. En el menú contextual, haga clic en **Generar solución**.
4. En el menú **Depurar**, haga clic en **Iniciar**. Compruebe que el proyecto Puzzle está definido como proyecto de inicio.

## Comentarios

Una vez que el proyecto se ha generado correctamente, intente resolver el puzzle.

Las características de guardado y carga del juego no están implementadas.

Para tener acceso al componente COM nativo desde los objetos administrados de .NET Framework, el ejemplo utiliza TLBIMP.EXE para generar una dll de proxy de .NET Framework.

## Clases del ejemplo

Las clases siguientes se utilizan este ejemplo.

Clase	Descripción
System.Windows.Forms.Form	Implementa el objeto <code>AboutForm</code> del proyecto Puzzle.
System.Object	Implementa el objeto <code>GameLevelEnum</code> del proyecto Puzzle.
System.Delegate	Implementa el objeto <code>SolveThreadProcDlg</code> del proyecto Puzzle.

## Vea también

[Ejemplos basados en varios lenguajes](#)

# Ejemplos generales

Los ejemplos de esta categoría son:

Ejemplo	Descripción
<a href="#">AutoWord</a>	Explica cómo automatizar Word desde Visual J#.
<a href="#">CarsSelector</a>	Muestra cómo crear una página de formularios Web Forms para seleccionar automóviles que simula una sencilla página de comercio electrónico. En este ejemplo se utiliza la tecnología ASP.NET.
<a href="#">LocalizedHelloWorld</a>	Explica cómo crear y utilizar ensamblados satélite para aplicaciones de consola.
<a href="#">MultilingualForm</a>	Muestra cómo crear y localizar formularios Windows Forms.
<a href="#">PerfMon</a>	Muestra cómo supervisar el rendimiento del sistema utilizando contadores de rendimiento, siendo también una introducción a los mismos.
<a href="#">ProcessController</a>	Explica cómo generar una aplicación para Windows en Visual J# que supervise procesos y servicios.
<a href="#">Scribble</a>	Muestra una aplicación de dibujo MDI de Windows de formularios Windows Forms.

**Vea también**

[Ejemplos de Visual J#](#) | [Ejemplos de aplicaciones](#) | [Ejemplos de tecnología](#) | [Ejemplos basados en varios lenguajes](#)

# Ejemplo AutoWord (automatización de Microsoft Word)

Este ejemplo incluye cuatro ejemplos que muestran cómo automatizar Microsoft Word desde Visual J#. En los ejemplos se explica cómo automatizar las siguientes actividades:

- Iniciar Word.
- Crear un documento nuevo.
- Abrir un documento existente.
- Enlazar eventos en una aplicación Word.

En el ejemplo también se muestra cómo crear un ensamblado Interop para Microsoft Word y algunas técnicas sencillas para automatizar Word.

## Proyectos del ejemplo

Este ejemplo contiene cinco proyectos y genera tres ensamblados.

## Generar y ejecutar el ejemplo

### Para crear los ensamblados

1. Abra la solución AutoWord.sln.
2. Haga clic con el botón secundario en el proyecto **RunCreateWordAssembly** y, a continuación, haga clic en **Generar**.
3. Para cada uno de los proyectos de ejemplo, haga clic con el botón secundario en el elemento **Referencias** y, a continuación, haga clic en **Agregar referencia**.
4. Haga clic en el botón **Examinar**, seleccione WORD.DLL en la lista de archivos .dll y, a continuación, haga clic en el botón **Abrir**.
5. Haga clic en el botón **Aceptar** para cerrar el explorador de referencias y haga clic en el botón **Aceptar** de la ventana emergente donde se indica que el directorio en el que se encontró WORD.DLL se agregará a la ruta de acceso de referencias.
6. Si tiene instalado Microsoft Office XP, agregue OFFICEXP a la lista de constantes de compilación. Para cada uno de los proyectos de ejemplo, haga clic con el botón secundario en la raíz del proyecto y, a continuación, haga clic en **Propiedades**.  
  
Aparece el cuadro de diálogo **Páginas de propiedades**.
7. Seleccione **Propiedades de configuración**, a continuación **Generar** y, finalmente, agregue **OFFICEXP** a la lista de **Constantes de compilación condicional**. Haga clic en **Aceptar**.
8. En el menú **Generar**, haga clic en **Generar solución**.

### Para ejecutar los ejemplos de automatización

1. Haga clic con el botón secundario en **Example1** en el **Explorador de soluciones** y elija **Establecer como proyecto de inicio** en el menú contextual. A continuación, genere y ejecute el proyecto.

En este ejemplo se muestra cómo crear una instancia del objeto de Word sin documentos.

2. Haga clic con el botón secundario en **Example2** en el **Explorador de soluciones** y elija **Establecer como proyecto de inicio** en el menú contextual. A continuación, genere y ejecute el proyecto.

En este ejemplo se explica cómo crear un nuevo documento Word y cómo incluir texto en el mismo.

3. Haga clic con el botón secundario en **Example3** en el **Explorador de soluciones** y elija **Establecer como proyecto de inicio** en el menú contextual. A continuación, genere y ejecute el proyecto.

En este ejemplo se muestra cómo abrir un documento Word existente y cómo modificarlo.

4. Haga clic con el botón secundario en **Example4** en el **Explorador de soluciones** y elija **Establecer como proyecto de inicio** en el menú contextual. A continuación, genere y ejecute el proyecto.

En este ejemplo se explica cómo enlazar varios eventos en Word.

## Clases y palabras clave del ejemplo

Este ejemplo muestra las siguientes clases:

- **System.Diagnostics.Process**
- **System.Environment**
- **System.Reflection.Missing**
- **System.Text.StringBuilder**
- **System.Windows.Forms.MessageBox**
- **System.Thread.Sleep**
- **System.Reflection.Missing.Value**
- **Microsoft.Win32.RegistryKey**
- **Word.Application**
- **Word.ApplicationEvents2\_DocumentChangeEventHandler**
- **Word.ApplicationEvents2\_DocumentOpenEventHandler**
- **Word.ApplicationEvents2\_NewDocumentEventHandler**
- **Word.ApplicationEvents2\_StartupEventHandler**
- **Word.Documents**
- **Word.\_Document**
- **Word.Range**
- **Word.Words**

**Vea también**

[Ejemplos generales](#)



# Ejemplo CarsSelector (crear y utilizar páginas Web de ASP.NET)

Este ejemplo muestra cómo generar una página de formularios Web Forms para seleccionar automóviles que simula una página sencilla de comercio electrónico. En el ejemplo se muestran las siguientes características:

- Enlace de datos: en este ejemplo se utilizan varios tipos de enlaces de datos:
  - Enlace de datos sencillo.
  - Enlace con una tabla de datos creada mediante programación (el origen de los datos no es una base de datos).
- Utilizar el controlador de eventos para modificar dinámicamente el enlace de varios controles.
- Utilizar controles de validación.
- Utilizar el control Calendar y validar sus entradas sin utilizar controles de validación.

## Requisitos

Para generar y ejecutar este ejemplo, necesita un equipo con los Servicios de Internet Information Server 5.0 con las Extensiones de servidor de FrontPage instaladas.

## Generar y ejecutar el ejemplo

### Para crear la solución

1. Copie los archivos de ejemplo en el disco duro.
2. Cree una nueva **Aplicación Web** y asígnele el nombre CarsSelector.
3. Elimine la página de formulario Web Forms predeterminada haciendo clic con el botón secundario en el archivo **WebForm1.aspx** y, a continuación, haciendo clic en **Eliminar** en el menú contextual.
4. Agregue el archivo CarsSelector.aspx al proyecto. Para ello, haga clic con el botón secundario en el nombre del proyecto, señale **Agregar** en el menú contextual y haga clic en **Agregar elemento existente**. En **Tipo de archivo**, seleccione Todos los archivos (\*.\*) y, a continuación, busque el archivo CarsSelector.aspx, selecciónelo y haga clic en **Abrir**.
5. Agregue el archivo CarsSelector.aspx.jsl al proyecto. Para ello, haga clic con el botón secundario en el nombre del proyecto, señale **Agregar** en el menú contextual y haga clic en **Agregar elemento existente**. A continuación, busque el archivo CarsSelector.aspx.jsl, selecciónelo y haga clic en **Abrir**. Haga clic en **Aceptar** para sobrescribir el archivo existente.
6. Establezca la página de formularios Web Forms como página de inicio. Para ello, haga clic con el botón secundario en el archivo **CarsSelector.aspx** y elija **Establecer como página de inicio** en el menú contextual.

Para obtener más información sobre cómo crear aplicaciones para formularios Web Forms, vea [Crear y administrar formularios Web Forms](#).

### Para generar y ejecutar este ejemplo

1. En el menú **Generar**, haga clic en **Generar solución**.
2. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Se mostrará la página de formularios Web Forms. Puede seleccionar un automóvil y elegir distintas opciones. El resultado aparecerá asociado a una etiqueta.

### Para utilizar el formulario Web Forms

1. Primero seleccione una marca de automóvil en la lista desplegable **Marca**.
2. Seleccione un modelo en la lista desplegable **Modelo**. Verá que el contenido de la lista depende de la marca de automóvil seleccionada.
3. Seleccione el **Año**, el **Color** y el **Kilometraje** en la lista desplegable correspondiente.
4. Escriba una dirección de correo electrónico en el cuadro de texto **Correo electrónico**.
5. Active una o varias casillas de verificación en **Características deseadas**.
6. Seleccione una fecha válida en el calendario **Fecha disponibilidad**.
7. Haga clic en **Enviar**.

## Clases y palabras clave del ejemplo

Este ejemplo muestra las siguientes clases y palabras clave:

- **System.Web.UI.Page**

- **DropDownList**
- **AutoPostBack**
- **CheckBoxList**
- **RequiredFieldValidator**
- **RegularExpressionValidator**
- **Dilatable**

**Vea también**

[Ejemplos generales](#)

# Ejemplo LocalizedHelloWorld (ensamblados satélite para aplicaciones de consola)

Este ejemplo explica cómo crear y utilizar ensamblados satélite para aplicaciones de consola. Muestra cómo crear recursos para aplicaciones de consola de Visual J#, localizar los recursos y generar ensamblados satélite localizados.

Aunque el código de este ejemplo está escrito en Visual J#, se pueden utilizar los principios y técnicas del ejemplo para localizar una aplicación administrada escrita en cualquier lenguaje.

Este ejemplo se divide en los siguientes proyectos y ensamblados:

Proyecto de utilidades de C++ Resources

Éste es el primer proyecto. Genera los recursos administrados con un paso de generación personalizada que llama a ResGen.exe para compilar los archivos .txt en archivos .resources. El archivo de texto de recursos contiene instrucciones <nombre>=<valor> y líneas de comentario que comienzan con el signo de almohadilla (#). Las cadenas se cargarán en el código con el nombre dado en el archivo.

Para los recursos en coreano, el archivo debe guardarse como UTF-8 para conservar los caracteres no ASCII. Observe también que no es necesario que cada referencia cultural tenga un valor para cada nombre.

Si un valor no existe para una referencia cultural en particular, el administrador de recursos recurre a las referencias culturales primarias. Por ejemplo, si la referencia cultural actual es ko-KR, el administrador de recursos buscará el nombre primero en los recursos "ko-KR", después en los recursos "ko" y por último en los recursos predeterminados. Desde el punto de vista técnico, los archivos de recursos pueden tener cualquier nombre, pero normalmente se utiliza la siguiente convención para nombrarlos:

```
<nombrebase>[.<referencia cultural>].resources
```

Si no se incluye ninguna referencia cultural, se utilizará la predeterminada.

Proyecto de consola HelloWorld de Visual J#

Éste es el segundo proyecto. Genera la aplicación principal de Visual J# e incrusta los recursos predeterminados (MyResources.resources, generado a partir MyResources.txt en el primer proyecto).

Proyecto de utilidades de C++ SatelliteAssemblies

Éste es el tercer proyecto. Genera los dos ensamblados satélite que contienen los recursos localizados con pasos de generación personalizada. Llama a Al.exe para crear el ensamblado satélite, pasando el archivo de recursos con la opción "/embed", estableciendo la referencia cultural con la opción "/culture" y copiando la información de ensamblado (título, versión, etc.) desde el ensamblado principal con la opción "/template".

Observe que los ensamblados satélite deben residir en un subdirectorio del ensamblado principal con el nombre de la referencia cultural, y que el nombre del archivo es el mismo que el del ensamblado principal pero con la palabra ".resources" antes de la extensión de archivo. Por ejemplo, si el ensamblado principal era "c:\bin\a.dll", el ensamblado satélite en en-US sería "c:\bin\en-US\a.resources.dll".

Observe también que el segundo argumento de la opción "/embed" es el nombre de los recursos. Éste es el nombre que utilizará el administrador de recursos para encontrar el grupo de recursos correspondiente y debe incluir la cadena de referencia cultural apropiada (distingue mayúsculas de minúsculas) después del nombre del archivo y antes de ".resources". Como Al.exe obtiene del ensamblado principal la mayor parte de información, éste proyecto debe ser generado después del segundo proyecto.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

1. Abra la solución LocalizedHelloWorld.sln.
2. En el menú **Generar**, haga clic en **Generar solución**.
3. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.
4. Escriba con una referencia cultural compatible con la aplicación para ver los recursos de dicha referencia cultural (el ejemplo incluye recursos en **en**, **en-US**, **ko-KR** y **predeterminado**, cuyo idioma es inglés).

### Para agregar una nueva referencia cultural

El siguiente procedimiento agrega recursos en inglés (Reino Unido).

1. Agregue un nuevo archivo de texto denominado "MyResources.en-GB.txt" al proyecto de utilidades de C++ Resources.
2. Agregue líneas <nombre>=<valor> válidas al nuevo archivo de texto, así como los comentarios deseados.
3. Agregue un paso de generación personalizada para el nuevo archivo (debe ser el mismo que el de los otros archivos de recursos del proyecto): ResGen.exe "\$ (InputPath)".
4. Establezca el resultado del paso de generación personalizada en \$(InputName).resources.
5. Genere el proyecto (no es necesario volver a generar el ensamblado principal).
6. Agregue el archivo existente denominado "..\Resources\MyResources.en-GB.resources" al proyecto de utilidades de C++ SatelliteAssemblies.
7. Agregue un paso de generación personalizada para este archivo (similar a los pasos de generación personalizada del resto de archivos):  
  

```
mkdir $(ProjectDir)\bin\$(OutDir)\en-GB

al.exe /culture:en-GB /out:"$(ProjectDir)\bin\$(OutDir)\en-GB\HelloWorld.resources.dll" /embed:"$(InputPath)","HelloWorld.Resources.$(InputFileName)" /template:"$(ProjectDir)\obj\$(IntDir)\HelloWorld.exe"
```
8. Establezca el resultado del paso de generación personalizada en "\$(ProjectDir)\bin\\$(OutDir)\en-GB\HelloWorld.resources.dll".
9. Genere el proyecto (no es necesario volver a generar el ensamblado principal).
10. Ejecute el código del ejemplo y pruebe el nuevo recurso.

### Para generar este ejemplo desde la línea de comandos

1. Compile el recurso para todas las referencias culturales (desde el subdirectorio Recursos):
  - ResGen.exe MyResources.txt

- ResGen.exe MyResources.en-US.txt
- ResGen.exe MyResources.ko-KR.txt

2. Compile el ensamblado principal e incruste los recursos predeterminados (desde el directorio raíz del proyecto):

```
vjc.exe /out:bin\Debug\HelloWorld.exe /res:Resources\MyResources.resources,HelloWorld.Resources.MyResources.resources HelloWorld.jsl AssemblyInfo.jsl
```

3. Genere los ensamblados satélite (desde el directorio raíz del proyecto):

```
mkdir bin\Debug\en-US
```

```
mkdir bin\Debug\ko-KR
```

```
al.exe /culture:en-US /out:bin\Debug\en-US\HelloWorld.resources.dll /embed:Resources\MyResources.en-US.txt,HelloWorld.Resources.MyResources.en-US.resources /template:bin\Debug\HelloWorld.exe
```

```
al.exe /culture:ko-KR /out:bin\Debug\ko-KR\HelloWorld.resources.dll /embed:Resources\MyResources.ko-KR.txt,HelloWorld.Resources.MyResources.ko-KR.resources /template:bin\Debug\HelloWorld.exe
```

**Nota** Puede agregar opciones de depuración o de cualquier otro tipo en la línea de comandos de Visual J#. También puede cambiar el directorio de resultados de "bin\Debug" a otro diferente.

## Vea también

[Ejemplos generales](#)

# Ejemplo MultilingualForm (crear y localizar formularios Windows Forms)

Este ejemplo muestra cómo crear y localizar formularios Windows Forms. Explica cómo utilizar el diseñador de Windows Forms para crear, modificar y generar aplicaciones localizadas.

Para obtener más información sobre aplicaciones localizadas, vea [Globalizar y localizar](#).

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

1. Abra la solución MultilingualForm.sln.
2. En el menú **Generar**, haga clic en **Generar solución**.
3. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.
4. Para ver recursos de otra referencia cultural, escriba un nombre de referencia cultural válido de RFC 1766 en el cuadro de texto y haga clic en el botón **modificar**. (Este ejemplo sólo incluye recursos para "en-US" y "ko-KR").

### Para localizar formularios Windows Forms y formularios Web Forms

1. Seleccione el formulario que desea localizar en el diseñador de formularios correspondiente.
2. Establezca la propiedad **Localizable** del formulario en true.
3. Haga lo siguiente para cada idioma al que desee localizar:
  - a. Establezca la propiedad **Language** del formulario en el idioma deseado.
  - b. Escriba el texto de cada control en el nuevo idioma.

También puede modificar el aspecto de los controles del formulario (tamaño, posición, etc.) para cada idioma.

## Vea también

[Ejemplos generales](#)

- Incluir nuevas clases de Visual J# derivadas de otros componentes.
- Agregar y quitar dinámicamente controladores de eventos y componentes.
- Utilizar el control **TreeView**, la clase **TreeNode** y el componente **Timer**.
- Utilizar los componentes **PerformanceCounterCategory** y **PerformanceCounter**.

**Nota** El código de este ejemplo sólo se puede ejecutar en Windows NT 4.0 y Windows 2000 y versiones posteriores.

### Para generar y ejecutar este ejemplo

- Se muestra la ventana principal de la aplicación.

## Supervisor de rendimiento



En cada pantalla gráfica de la ilustración se muestra lo siguiente:

- En la parte superior, el nombre del contador y el nombre de la instancia separados por "\*\*\*\*". Si la categoría no contiene un nombre de instancia (por ejemplo, la categoría ICMP), la cadena "\*\*\*\*" es seguida por un espacio en blanco.
- En la parte inferior, el valor actual del contador. Dicho valor puede cambiar periódicamente (la frecuencia predeterminada es cada segundo).
- En el lado derecho, una barra roja. El alto de la barra depende del último valor supervisado en el contador.
- En el gráfico, la historia del rendimiento del contador.

### Para cargar los contadores de rendimiento de un sistema

Ésta es una aplicación MDI. Al iniciarse, aparecen cuatro ventanas que se pueden utilizar para supervisar cuatro equipos. Puede hacer clic en el comando **Nuevo PerfMon** del menú **Opciones** para abrir tantas ventanas como desee. En los procedimientos siguientes sólo se utiliza una ventana. Por tanto, puede cerrar todas las ventanas excepto una.

1. En el campo **Nombre del equipo**, escriba el nombre del sistema que desea supervisar. Si lo deja en blanco, se supervisará el equipo local.
2. Haga clic en el botón **Cargar contadores** (o presione Alt+L). La aplicación cargará los componentes **PerformanceCounterCategory**, **InstanceName** y **PerformanceCounter** en el equipo especificado. El botón **Cargar contadores** permanecerá deshabilitado hasta que se hayan cargado todos los contadores en el sistema.

La vista de árbol del panel izquierdo mostrará las categorías de rendimiento y los contadores de rendimiento disponibles. Cada hoja del árbol representa un contador de rendimiento. Puede supervisar el progreso completo observando el indicador de progreso en la parte inferior de la ventana de aplicación. Cuando se haya completado todo el árbol, se habilitará de nuevo el botón **Cargar contadores**.

**Nota** Para supervisar un equipo en concreto, necesita disponer de permisos de administrador en dicho equipo. En caso contrario, los componentes **PerformanceCounterCategory** no podrán cargar las categorías (protección de seguridad).

### Para supervisar el rendimiento de un sistema

1. Cargue los contadores de rendimiento siguiendo los pasos anteriores.
2. Seleccione uno de los contadores expandiendo el nodo de la categoría correspondiente. Por ejemplo, para observar el rendimiento del procesador, expanda el nodo **Processor** y, a continuación, seleccione una de las instancias de **Processor** (por ejemplo, "\_Total" o "0"). Expanda también el nodo de la instancia para ver los contadores.
3. En el nivel de hoja, los contadores se representan mediante casillas de verificación. Seleccione uno de los contadores, por ejemplo, **%Processor Time**. Aparecerá una ficha llamada **Processor** en el panel derecho, que mostrará una pantalla gráfica del contador seleccionado.
4. Para quitar una pantalla gráfica determinada de la ficha, desactive la casilla de verificación del contador correspondiente. Cuando se desactivan todos los contadores de una misma página, desaparece la ficha correspondiente.

### Para administrar las propiedades de las pantallas

**Nota** Cada vez que cambie las propiedades de la pantalla, haga clic en el botón **Actualizar pantalla** para aplicar los cambios. Los cambios tendrán efecto en la pantalla actual y en las siguientes.

- Para modificar el grosor de línea, seleccione el grosor deseado en el cuadro **Grosor de línea**.
- Para modificar el número de filas por pantalla, seleccione el número deseado en el cuadro **Filas**. Por ejemplo, si selecciona una fila, la pantalla ocupará todo el alto de la página de la ficha.
- Para modificar el número de columnas por pantalla, seleccione el número deseado en el cuadro **Columnas**. Por ejemplo, si selecciona una columna, la pantalla ocupará todo el ancho de la página de la ficha.

**Nota** Si el número de pantallas gráficas excede el área visible de la página de la ficha, se crearán automáticamente barras de desplazamiento.

- Para modificar el intervalo de tiempo, seleccione el número de segundos en el cuadro **Frecuencia**.
- Para modificar el número de puntos del gráfico, seleccione un número en el cuadro **Densidad**.

### Para combinar dos o más pantallas

1. Arrastre la pantalla que desea mover y colóquela en la pantalla destino. Observe que el fondo de la pantalla destino cambia a negro al seleccionarla para recibir la pantalla que se está moviendo. Al combinarse ambas pantallas, aparecerán dos

colores de gráfico diferentes (en este caso rojo y verde). Por ejemplo, si coloca "%Processor Time \*\*\* \_total" en "%User Time \*\*\* \_total", el color verde representará a la primera y el rojo a la segunda. Los datos mostrados en la parte superior de la pantalla, en la parte inferior y en el lado derecho, representarán sólo al contador "%Processor Time \*\*\* \_total".

2. Para cambiar de un gráfico a otro después de una combinación de pantallas, haga clic con el botón secundario del *mouse* (ratón) en la pantalla y seleccione el nombre del gráfico en el menú contextual. En el ejemplo anterior, si desea cambiar los datos de la pantalla de "% Processor Time \*\*\* \_total" a "%User Time \*\*\* \_total", haga clic con el botón secundario del *mouse* en la pantalla y seleccione "%User Time \*\*\* \_total" en el menú contextual. La pantalla mostrará los datos del contador seleccionado y el color de la barra del lado derecho cambiará a rojo (o al color que represente al contador seleccionado).
3. Si existen más de dos contadores combinados, cada contador estará representado por un color diferente. Para seleccionar la vista de datos para cualquiera de los contadores, realice el paso anterior.

### Para separar dos o más pantallas combinadas

1. Busque la casilla de verificación del contador de la pantalla que desea separar.
2. Desactive la casilla de verificación del contador. Se eliminará la pantalla correspondiente.
3. Active de nuevo la casilla de verificación del contador. Aparecerá una nueva pantalla en la página de la ficha.

### Para utilizar las opciones de menú

- Para detener el trazado de gráficos:
  1. Haga clic en el menú **Opción** (o presione Alt+O).
  2. Haga clic en **Detener gráficos**.

Los gráficos quedarán congelados y ya no se trazarán más líneas.

- Para reiniciar el trazado de gráficos:
  1. Haga clic en el menú **Opción**.
  2. Haga clic en **Iniciar gráficos**.

Las pantallas continuarán el trazado desde la última posición.

- Para salir de la aplicación:
  1. Haga clic en el menú **Opción**.
  2. Haga clic en **Salir**.

### Vea también

[Ejemplos generales](#)



# Ejemplo ProcessController (supervisión de procesos y servicios)

Este ejemplo explica cómo generar una aplicación para Windows en Visual J# que supervise procesos y servicios. También se explican las actividades siguientes:

- Agregar y quitar dinámicamente controladores de eventos.
- Utilizar el componente **Process** para:
  - Leer los procesos que se están ejecutando en un equipo local o remoto.
  - Iniciar un proceso en un equipo local.
  - Detener un proceso en un equipo local o remoto.
  - Leer los módulos que utiliza un proceso en ejecución.
  - Leer todas las características de un proceso en ejecución.
- Utilizar el componente **ServiceController** para:
  - Leer los servicios en ejecución y los controladores de dispositivo de un equipo local o remoto.
  - Iniciar, detener o pausar un servicio de un equipo local o remoto.
  - Comprobar las propiedades de los servicios o de los controladores de dispositivo de un equipo local o remoto.
- Controlar problemas de seguridad relacionados con el uso de los componentes **Process** y **ServiceController**.

**Nota** El código de este ejemplo sólo puede ejecutarse en los sistemas operativos Windows NT o Windows 2000 ya que el componente **ServiceController** sólo es compatible con plataformas Windows NT.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

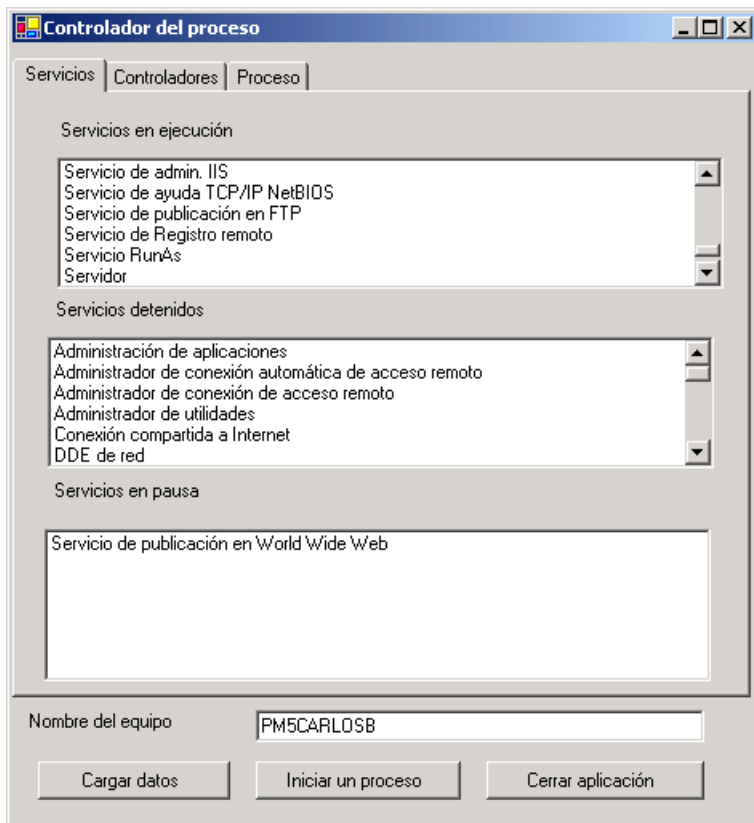
1. Abra la solución ProcessController.sln.
2. En el menú **Generar**, haga clic en **Generar solución**.
3. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Se muestra la ventana principal de la aplicación.

La siguiente ilustración muestra cómo se ejecuta el programa ProcessController. Puede ir consultando la ilustración a medida que vaya realizando los siguientes procedimientos de prueba:

- Cargar todos los servicios, controladores de dispositivo y procesos en ejecución de algún equipo de la red.
- Iniciar, detener o pausar un servicio en ejecución o un controlador de dispositivo de un equipo local o remoto.
- Iniciar un proceso en ejecución de un equipo local.
- Terminar un proceso en ejecución de un equipo local o remoto.

### Controlador de procesos



### Para cargar todos los servicios, controladores de dispositivo y procesos

1. Escriba el nombre del equipo deseado en el cuadro **Nombre del equipo**.
2. Haga clic en el botón **Cargar datos** (o presione Alt+L).

### Para iniciar un servicio

1. Haga clic en un nombre de servicio de la lista **Servicios detenidos**.
2. Haga clic con el botón secundario en el servicio y, a continuación, haga clic en **Iniciar servicio** en el menú contextual.

Una vez iniciado el servicio, su nombre aparecerá en la lista **Servicios en ejecución** de la parte superior.

**Nota** No es posible iniciar un servicio en un equipo si no dispone de permisos de administrador en dicho equipo.

### Para detener un servicio

1. Haga clic en un nombre de servicio de la lista **Servicios en ejecución**.
2. Haga clic con el botón secundario en el servicio y, a continuación, haga clic en **Detener servicio** en el menú contextual.

Una vez detenido el servicio, su nombre aparecerá en la lista **Servicios detenidos**.

**Nota** Algunos servicios no admiten el estado **Detenido**. Tampoco puede detener un servicio si no dispone de permisos de administrador en el equipo.

### Para pausar un servicio en ejecución

1. Haga clic en un nombre de servicio de la lista **Servicios en ejecución**.
2. Haga clic con el botón secundario del *mouse* en el servicio y, a continuación, haga clic en **Pausa** en el menú contextual.

Si el servicio está en pausa, el nombre del servicio aparecerá en la lista **Servicios pausados** de la parte inferior de la ventana.

**Nota** Algunos servicios no admiten el estado **Pausado**. Tampoco puede pausar un servicio si no dispone de permisos de administrador en el equipo.

### Para comprobar información de un servicio

1. Haga clic en un nombre de servicio de cualquiera de las tres listas.
2. Haga clic con el botón secundario en el servicio y, a continuación, haga clic en **Mostrar información del servicio** en el menú contextual. Se muestra una nueva ventana que contiene información sobre el servicio.

Para comprobar y administrar controladores de dispositivo

- 1. Haga clic en la ficha **Controladores** en la parte superior de la ventana.
- 2. Siga los pasos explicados anteriormente para iniciar, detener y pausar un servicio. Las reglas mencionadas para los servicios se aplican igualmente a los controladores de dispositivo.

Para comprobar y administrar procesos

- 1. Haga clic en la ficha **Procesos** de la parte superior de la ventana.
- 2. Seleccione un nombre de proceso en la lista del extremo izquierdo. En la lista del lado derecho aparecerá información sobre el servicio.

**Nota** Si no tiene permisos de administrador en el equipo destino, no podrá ver o administrar ningún proceso.

Para iniciar un proceso

- 1. Haga clic en el botón **Iniciar proceso** (o presione Alt+S).
- 2. Seleccione el proceso que desea iniciar en el cuadro de diálogo **Abrir**.
- 3. Haga clic en el botón **Abrir** del cuadro de diálogo. Si el equipo destino es el equipo local, el nombre del proceso aparecerá en la lista del extremo izquierdo.

**Nota** Si el equipo destino es un equipo remoto, el proceso se iniciará de manera predeterminada en el equipo local, si esto es posible.

Para detener un proceso

- 1. Haga clic en un proceso de la lista del extremo izquierdo.
- 2. Haga clic con el botón secundario en el proceso y, a continuación, haga clic en **Terminar proceso** en el menú contextual.

Una vez terminado el proceso, su nombre desaparecerá de la lista.

Para cargar servicios, controladores de dispositivo y procesos de otro equipo

- 1. Escriba el nombre del equipo en el cuadro **Nombre del equipo**.
- 2. Haga clic en el botón **Cargar datos** (o presione Alt+L).

Para detener la aplicación ProcessController

- Haga clic en el botón **Cerrar aplicación** (o presione Alt+C).

La ventana se irá atenuando lentamente. Este efecto se obtiene mediante la propiedad **Opacity** de la ventana.

Clases del ejemplo

Las clases siguientes se utilizan este ejemplo.

Archivo	Descripción
MainForm.jsl	Describe la clase <b>MainForm</b> , que es una clase de tipo <b>System.Windows.Forms.Form</b> . El diseñador contiene la mayoría de los controles de interfaz de usuario necesarios para esta aplicación.
ProcessControllerManager.jsl	Contiene la implementación de la clase <b>ProcessControllerManager</b> . Esta clase controla la administración de procesos.
DriverControllerManager.jsl	Contiene la implementación de la clase <b>DriverControllerManager</b> . Esta clase se encarga de administrar los controladores de dispositivo de un equipo.
ServiceControllerManager.jsl	Contiene la implementación de la clase <b>ServiceControllerManager</b> . Esta clase describe la lógica para controlar los servicios de un equipo.
ServiceInfo.jsl	Implementa la clase <b>ServiceInfo</b> , que es una clase <b>System.Windows.Forms.Form</b> y que se utiliza sólo para mostrar información de servicios y controladores.

Vea también

[Ejemplos generales](#)

# Ejemplo Scribble (aplicación de dibujo MDI)

Este ejemplo muestra cómo desarrollar una aplicación MDI de formularios Windows Forms utilizando clases de Visual J# y de .NET Framework. Scribble es una sencilla aplicación de dibujo que le permite dibujar a mano alzada mediante el *mouse* y guardar las imágenes en un archivo.

## Generar y ejecutar el ejemplo

### Para generar y ejecutar este ejemplo

1. Abra la solución Scribble.sln.
2. En el menú **Generar**, haga clic en **Generar solución**.
3. En el menú **Depurar**, haga clic en **Iniciar sin depurar**.

Cuando ejecute el código del ejemplo, obtendrá una superficie de dibujo y los siguientes elementos de menú:

- Menú **Archivo** para operaciones de entrada y salida.
- Menú **Edición** para cortar, copiar y pegar.
- Menú **Lápiz** para seleccionar el grosor del lápiz.
- Menú **Ver** para mostrar u ocultar la barra de herramientas y la barra de estado.
- Menú **Ventana** para crear o disponer las ventanas en mosaico o en cascada.
- Menú **Ayuda** para mostrar los temas de ayuda y el cuadro de texto "Acerca de Scribble".

Además de estas opciones de menú, la interfaz de usuario incluye una barra de herramientas con botones para duplicar los comandos de menú: **Nuevo**, **Abrir**, **Guardar**, **Vista preliminar**, **Imprimir** y **Ayuda**.

## Otras versiones de Scribble

Scribble también está disponible como ejemplo de MFC, como ejemplo de extensiones administradas de C++ y como ejemplo de Visual Basic:

- [Ejemplo Scribble: Aplicación de dibujo MDI de MFC](#)
- [Ejemplo Scribble: aplicación de dibujo MDI mediante Extensiones administradas de C++](#)
- [Windows Forms: ejemplo Scribble](#)

## Vea también

[Ejemplos generales](#)

# Ejemplos de tecnología

Los ejemplos de esta sección muestran las distintas tecnologías y características de .NET Framework y Visual J#. La intención es explicar una única tecnología o varias tecnologías relacionadas. Estos ejemplos no son estrictamente útiles como aplicaciones completas y tienden a mostrar control de errores y desarrollo orientado a objetos limitados:

Categoría	Ejemplo
Cargador de clases	<a href="#">Ejemplo ClassLoader</a> (modificar un cargador de clases personalizado para ensamblados)
CLS Extender	<a href="#">Ejemplo CLSExtender</a> (crear propiedades, eventos y delegados de .NET Framework)
CodeDOM	<a href="#">Ejemplo CodeDOM</a> (utilizar CodeDomProvider)
Entre lenguajes	<a href="#">Ejemplo CrossLanguage</a> (interoperabilidad de Visual J# con otros lenguajes administrados)
Carga dinámica	<a href="#">Ejemplo DynamicLoading</a> (cargar clases de ensamblados implementados)
Java y COM	<a href="#">Ejemplo COMCallsJava</a> (actualizar componentes de Java de Visual J++ 6.0)
	<a href="#">Ejemplo JavaCallsCOM</a> (actualizar componentes de Java de Visual J++ 6.0)
Java y COM (utilizar semántica de .NET Framework)	<a href="#">Ejemplo COMCallsJava_NET</a> (exponer componentes administrados en COM)
	<a href="#">Ejemplo JavaCallsCOM_NET</a> (exponer objetos COM en componentes administrados)
Invocación de plataforma	<a href="#">Ejemplo PInvoke</a> (llamar a API no administradas)
Ámbito	<a href="#">Ejemplo Scoping</a> (utilizar /securescoping)
Proveedores de seguridad	<a href="#">Ejemplo SecurityProviders</a> (especificar proveedores de seguridad de otros fabricantes)
Uso de .NET Framework	<a href="#">Ejemplo Attributes</a> (utilizar atributos de .NET Framework)
	<a href="#">Ejemplo Enums</a> (utilizar enumeraciones)
	<a href="#">Ejemplo EventsAndDelegates</a> (utilizar eventos y delegados de .NET Framework)
	<a href="#">Ejemplo Minesweeper</a> (Generar una versión del juego Buscaminas)
	<a href="#">Ejemplo Properties</a> (utilizar propiedades de .NET Framework)
	<a href="#">Ejemplo ValueTypes</a> (utilizar tipos de valor de .NET Framework)
Servicio remoto	<a href="#">Ejemplo de servicio remoto</a>
Windows Forms y ampliación de aplicaciones WFC	<a href="#">Ejemplo WFCUsingWinForm</a> (ampliar una aplicación WFC generada en Visual J++ 6.0 con Windows Forms)

## Vea también

[Ejemplos de Visual J#](#) | [Ejemplos de aplicaciones](#) | [Ejemplos generales](#) | [Ejemplos basados en varios lenguajes](#)

# Ejemplo ClassLoader (modificar un cargador de clases personalizado para ensamblados)

Este ejemplo muestra cómo se pueden modificar los cargadores de clases personalizados de aplicaciones existentes de Java para cargar clases desde ensamblados administrados. Para el código nuevo, se recomienda buscar y cargar las clases desde ensamblados administrados utilizando la semántica de Common Language Runtime y las API de .NET Framework en lugar de utilizar cargadores de clases personalizados. Los cargadores de clases están pensados sólo para utilizarlos en aplicaciones de Java existentes. Los métodos **resolveClass** y **defineClass** de la clase **java.lang.ClassLoader** no son compatibles. En aplicaciones Java existentes, tendrá que escribir de nuevo el método **loadClass** para que llame a una de las versiones de **Class.forName** y crear las entradas correspondientes en el archivo de configuración (.config) de la aplicación para que pueda encontrar y cargar los ensamblados administrados necesarios. Si los ensamblados administrados se encuentran en el directorio de trabajo de la aplicación, no es preciso ningún archivo de configuración; la heurística de búsqueda **Class.forName** buscará la clase. Vea [Archivos de configuración de aplicaciones](#) para obtener más información sobre la sintaxis de los archivos de configuración (.config) de aplicaciones.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo ClassLoader.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo:

- En el entorno de desarrollo, presione F5 para generar y ejecutar el ejemplo.  
O bien
- En la línea de comandos, escriba **ClassLoaderSample.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo CLSExtender (crear propiedades, eventos y delegados de .NET Framework)

Este ejemplo muestra la creación de propiedades, eventos y delegados de .NET Framework.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo Container.sln y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo:

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **Container.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo CodeDOM (utilizar CodeDomProvider)

En este ejemplo se muestra el uso de CodeDomProvider de Visual J# para generar código y compilarlo de manera dinámica utilizando CodeDOM. En este ejemplo:

- Se genera el árbol codeDOM para un sencillo programa Hola a todos.
- Se genera código fuente para éste en el archivo bin\GeneratedFile.jsl.
- Se compila el archivo generado utilizando la implementación CodeDomProvider para Visual J# para dar el ensamblado generado "hello world!"

Para obtener más información sobre CodeDOM, vea [Referencia rápida de CodeDOM](#).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo integrado (IDE), abra CodeDOMUsage.sln y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5 para generar y ejecutar el ejemplo. El archivo generado GeneratedFile.jsl está también disponible.  
O bien
- En la línea de comandos, escriba **CodeDOMUsage.exe**. El archivo generado GeneratedFile.jsl está disponible en el directorio bin\debug.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)



# Ejemplo CrossLanguage (interoperabilidad de Visual J# con otros lenguajes administrados)

Este ejemplo muestra cómo utilizar la sintaxis de Java en Visual J# para tener acceso a clases escritas en otros lenguajes y cómo pueden tener acceso otros lenguajes a clases de Visual J#. También se muestra cómo pueden capturar otros lenguajes excepciones de Java iniciadas por componentes escritos en Java. En este ejemplo se crean tres ensamblados. Estos tres ensamblados de biblioteca o DLL definen clases sencillas escritas en Extensiones administradas de C++, Visual Basic y Visual J#. Estas clases derivan las unas de las otras y, en última instancia, de la clase base escrita en las Extensiones administradas de C++. Finalmente, el archivo ejecutable escrito en C# crea instancias de la clase de Visual J# y llama a sus métodos. .NET Framework es un entorno en el que varios programadores pueden trabajar juntos sin dificultades con el lenguaje que cada uno prefiera.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- Escriba **BUILD.bat** desde la línea de comandos.

**Nota** La ubicación de uno de los archivos DLL necesarios para compilar el ejemplo está codificada en la secuencia de comandos de compilación como C:\Windows\Microsoft .NET\Framework\v<%versión%>\vjslib.dll. Puede ser necesario cambiarla por la versión correcta de Visual J# instalada en el equipo y el nombre del directorio de Windows.

### Para ejecutar este ejemplo

- Escriba **CrossLang.exe** en la línea de comandos.

Para cada objeto del que se ha creado una instancia, el código al que se llama está escrito en al menos dos lenguajes de desarrollo diferentes.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo DynamicLoading (cargar clases de ensamblados implementados)

En este ejemplo se muestra cómo se pueden cargar clases de DLL administradas implementadas en varias ubicaciones utilizando las API de `Class.forName`. Hay dos versiones de **Class.forName**:

- **Class.forName(String nombreClase)**
- **Class.forName(String nombreEnsamblado, String nombreClase, boolean rutaAbsoluta)**

Vea [Enlaces en tiempo de ejecución: semántica de Class.forName\(\)](#) para obtener información detallada sobre estas API.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- Escriba **BUILD.bat** desde la línea de comandos.

### Para ejecutar este ejemplo

- Escriba **DynamicLoading.exe** en la línea de comandos.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar `vsvars.bat` (que se encuentra en el directorio `<%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools`). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio `<raíz de instalación de Microsoft Visual J++ 6.0>`.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo COMCallsJava (actualizar componentes de Java y COM de Visual J++ 6.0)

Este ejemplo muestra la actualización de componentes de Java de Visual J++ 6.0 expuestos como archivos DLL COM. Para actualizar el componente de Visual J++ 6.0, se genera una DLL de contenedor administrado a partir de la biblioteca de tipos del servidor COM utilizando la herramienta TlbImp de .NET Framework. A continuación, se hace referencia a esta DLL administrada al compilar los archivos de código fuente. La DLL emitida se registra después utilizando la herramienta RegAsm de .NET Framework.

Para obtener información detallada sobre la actualización de aplicaciones de Java o COM de Visual J++ 6.0 a Visual J#, vea [Actualizar componentes de Visual J++ 6.0 expuestos a COM](#).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, vaya al directorio JavaServer, abra el archivo JavaServer.vjsproj y genere el servidor (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat** en el directorio raíz del ejemplo.

### Para ejecutar este ejemplo

- Vaya al directorio COMClient y abra el archivo COMClient.vbp en Visual Basic® 6.0.  
O bien
1. En el menú **Proyecto**, elija **Referencias**.
  2. Agregue una referencia al archivo Server.tlb del directorio ..\JavaServer\. Genere y ejecute la demostración presionando F5.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo JavaCallsCOM (actualizar componentes de Java y COM de Visual J++ 6.0)

En este ejemplo se muestra la actualización de aplicaciones de Java o COM de Visual J++ 6.0 a Visual J#. En el ejemplo hay tres directorios dentro del subdirectorio Clients. El cliente del directorio SimpleClient muestra la actualización de un cliente sencillo que tiene acceso a una DLL COM. Incluso se actualizan los contenedores generados por JActiveX®. Los ejemplos InterfaceMarshaling y MessagePump explican soluciones específicas para problemas relacionados con el subprocesamiento que pueden encontrarse los desarrolladores al actualizar sus aplicaciones a Visual J#.

Para obtener información detallada sobre la actualización de aplicaciones de Java o COM de Visual J++ 6.0 a Visual J#, vea [Actualizar aplicaciones de Visual J++ 6.0 que tienen acceso a componentes COM](#).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo en el entorno de desarrollo

1. Abra el archivo COMServer.vcproj desde el directorio \COMServer y genere el servidor (Ctrl+Mayús+B).
2. En el directorio cliente pertinente, escriba **GenerateWrappers.bat**.
3. Abra el archivo de proyecto correspondiente (SimpleClient.vjsproj, InterfaceMarshaling.vjsproj o MessagePump.vjsproj) en el entorno de desarrollo y génerele.

### Para generar este ejemplo desde la línea de comandos

- Escriba **BUILDALL.bat** desde el directorio raíz del ejemplo.

Se incluye una DLL COM. Si no desea generar de nuevo el servidor COM, debe registrar la DLL existente escribiendo **regsvr32 COMServer.dll** para poder utilizarla. Si genera de nuevo el servidor COM, no es necesario el registro explícito, puesto que lo hace el proceso de generación.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5 para generar y ejecutar el ejemplo.  
O bien
- En la línea de comandos, escriba **Client.exe** desde el directorio cliente pertinente.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo COMCallsJava\_NET (exponer componentes administrados en COM utilizando la interoperabilidad de .NET Framework)

Este ejemplo muestra cómo se pueden exponer en COM componentes administrados escritos en Visual J# utilizando la semántica de interoperabilidad COM de .NET Framework. El uso de la semántica de .NET Framework es el modo recomendado para la interoperabilidad COM para los componentes de Visual J# recién escritos. En este ejemplo, el componente administrado escrito en Visual J# se compila como una DLL. A continuación, se registra la DLL como un componente COM utilizando la herramienta RegAsm, que permite que tengan acceso a ella clientes como Visual Basic 6.0.

Para obtener más información sobre la interoperabilidad COM utilizando la semántica de .NET Framework, vea [Interoperabilidad con código no administrado](#).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo Server.vjsproj y genere el servidor (Ctrl+Mayús+B).
- En la línea de comandos, escriba **BUILD.bat** en el directorio raíz del ejemplo.

### Para ejecutar este ejemplo

1. Abra el archivo VB6Project.vbp en Visual Basic 6.0.
2. En el menú **Proyecto**, elija **Referencias**. A continuación, busque y agregue una referencia al archivo MyForm.tlb en el directorio del ejemplo o en el subdirectorio .\bin\debug del directorio raíz del proyecto al generar en el IDE.
3. Presione F5.
4. Una vez en ejecución, haga clic en el botón de formulario de Visual Basic para crear un formulario administrado. Ajuste las propiedades en el formulario de Visual Basic y observe los cambios que se producen en el formulario administrado. El formulario de Visual Basic permite también crear un botón en el formulario administrado y ajustar algunas de sus propiedades. Todo esto se lleva a cabo desde código no administrado de Visual Basic.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo JavaCallsCOM\_NET (exponer objetos COM en componentes administrados utilizando la interoperabilidad de .NET Framework)

Este ejemplo muestra cómo pueden tener acceso las aplicaciones de Visual J# a DLL COM utilizando la semántica de interoperabilidad COM de .NET Framework. El uso de la semántica de .NET Framework es el modo recomendado para la interoperabilidad COM para las aplicaciones de Visual J# recién escritas. Los contenedores administrados para la DLL COM se generan utilizando la herramienta TlbImp de .NET Framework. Después, se importan al programa de Visual J#. El uso del operador **new** en los contenedores correspondientes crea una instancia de la coclase COM. El desarrollador puede invocar así a los métodos de la DLL COM llamando a los métodos del contenedor. A diferencia de la interoperabilidad mediante contenedores de JActiveX®, los contenedores generados con TlbImp proceden de System.Object y toman los tipos de datos de .NET Framework como parámetros y tipos de valor devuelto.

Para obtener más información sobre la interoperabilidad COM utilizando la semántica de .NET, vea

[Exponer componentes de .NET Framework en COM](#).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo COMServer.vcproj desde el directorio \COMServer y genere el servidor (Ctrl+Mayús+B). Desde el directorio \Client, abra el archivo Client.vjsproj y génerele.

O bien

- En la línea de comandos, escriba **BUILDALL.bat** en el directorio raíz del ejemplo.

Se incluye una DLL COM ya generada. Si no desea generar de nuevo el servidor COM, debe registrar la DLL existente escribiendo **regsvr32 COMServer.dll** antes de utilizarla. Si genera de nuevo el servidor COM, no es necesario el registro explícito, puesto que lo hace el proceso de generación.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, abra el archivo Client.vjsproj (desde el directorio \Client) y presione F5 para generar y ejecutar el ejemplo.

O bien

- En la línea de comandos, escriba **Client.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo PInvoke (llamar a API no administradas)

En este ejemplo se muestra cómo Visual J# puede utilizar los servicios de invocación de plataforma que proporciona Common Language Runtime para llamar a API no administradas. La invocación de plataforma es una alternativa a la tecnología J/Direct® (J/Direct es totalmente compatible con Visual J#), pero utiliza atributos de interoperabilidad y tipos de datos de .NET Framework para llamar a API no administradas, en lugar de las directivas **@dll()** y los tipos de Java que utiliza J/Direct.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo PInvoke.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo:

- En el entorno de desarrollo, presione F5 para generar y ejecutar el ejemplo.  
O bien
- En la línea de comandos, escriba **PInvokeSample.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo Scoping (utilizar /securescoping)

En este ejemplo se muestran las dos opciones de ámbito (utilizando /securescoping) disponibles en vjc.exe y las diferencias de semántica entre dos aplicaciones compiladas y ejecutadas en Visual J# que tienen diferente semántica de ámbito para miembros con ámbito protegido y de paquete. Estas diferencias afectan a los paquetes compilados en dos o más ensamblados y, de manera más significativa, a componentes escritos en Visual J# que se exponen a otros lenguajes en .NET. La opción de ámbito seguro se trata con detalle en [/securescoping](#).

En el ejemplo, una clase con ámbito de paquete en un ensamblado (un archivo .EXE) intenta tener acceso a otra clase con ámbito del mismo paquete pero en un ensamblado diferente (una .DLL). Cuando se compila la DLL sin la opción /securescoping, la clase del archivo .EXE puede tener acceso a la clase con ámbito de paquete de la DLL. Cuando se compila con la opción /securescoping, se produce un error java.lang.IllegalAccessException.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo Scoping.sln y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- Escriba **BUILD.bat** desde la línea de comandos.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- Escriba **PublicClass.exe** en la línea de comandos.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)



# Ejemplo SecurityProviders (especificar proveedores de seguridad de otros fabricantes)

Este ejemplo muestra cómo se puede utilizar el archivo `vjsharp.config` para especificar los proveedores de seguridad de otros fabricantes que se deben utilizar para llevar a cabo operaciones criptográficas como la generación de claves y firmas. El archivo `vjsharp.config` no se instala de manera predeterminada. Para utilizar proveedores de seguridad de otros fabricantes para operaciones criptográficas en lugar del proveedor de seguridad predeterminado que se distribuye con las bibliotecas de clases de Visual J#, el usuario debe crear este archivo y copiarlo en el directorio `<%windir%>\Microsoft .NET\Framework\v<%versión%>`. Vea [Cargar proveedores de seguridad de otros fabricantes](#) para obtener más información sobre el formato de este archivo y cómo se cargan proveedores de seguridad utilizando este archivo.

Este ejemplo crea un proveedor de seguridad ficticio que proporciona una implementación de mensaje implícita utilizando un algoritmo *Sample*. A continuación, se crea el archivo `vjsharp.config` para especificar que este proveedor se debe tener en consideración durante la búsqueda de clases implícitas de mensaje implementadas con el algoritmo *Sample*. Una aplicación intenta después cargar una clase del proveedor de seguridad ficticio utilizando la API **getInstance** en la clase **java.security.MessageDigest**.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- Escriba **BUILD.bat** en la línea de comandos y copie el archivo `vjsharp.config` del directorio raíz del ejemplo en `<%windir%>\Microsoft .NET\Framework\v<%versión%>`.

### Para ejecutar este ejemplo

- Escriba **SecurityProvider.exe** en la línea de comandos.

**Nota** Antes de ejecutar el ejemplo, actualice el archivo de configuración `vjsharp.config` asociado con la versión correcta del ensamblado y el símbolo de clave pública de `vjslib.dll` en el que se está ejecutando el ejemplo.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar `vsvars.bat` (que se encuentra en el directorio `<%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools`). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio `<raíz de instalación de Microsoft Visual J++ 6.0>`.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo Attributes (utilizar atributos de .NET Framework)

En este ejemplo se muestra cómo aplicaciones de lenguaje compiladas y ejecutadas en Visual J# pueden utilizar los atributos de .NET Framework para proporcionar funcionalidad mejorada. Todos los atributos disponibles en .NET Framework se pueden asociar a programas de Visual J#. Sin embargo, el compilador de Visual J# no admite la creación (definición) de atributos en los programas de Visual J#.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo Attributes.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **AttributeSample.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo Enums (utilizar enumeraciones)

En este ejemplo se muestra el uso de las enumeraciones de .NET Framework. Se pueden utilizar enumeraciones de .NET Framework desde Visual J#, pero no se permite crearlas (definirlas).

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo Enums.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **EnumSample.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

Para obtener más información sobre el uso de enumeraciones de .NET Framework, vea [Utilizar enumeraciones](#).

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo EventsAndDelegates (utilizar eventos y delegados de .NET Framework)

En este ejemplo se muestra el uso de los delegados y eventos de .NET Framework. Puede consumir delegados de .NET Framework en Visual J#. Las palabras clave **delegate** o **multicast** crean delegados de tipo **com.ms.lang.Delegate** en lugar de **System.Delegate**.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo EventsAndDelegates.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **EventsAndDelegates.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo Minesweeper (Generar una versión del juego Buscaminas)

Este ejemplo muestra cómo se pueden orientar aplicaciones a las siguientes características de .NET Framework y cómo pueden consumirlas:

- Eventos
- Delegados
- Tipos de valor
- Propiedades
- Atributos

Se utilizan las siguientes clases de .NET Framework de los siguientes espacios de nombres:

- **System**
- **System.Drawing**
- **System.Windows.Forms**
- **System.ComponentModel**

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo minesweeper.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **Minesweeper.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo Properties (utilizar propiedades de .NET Framework)

En este ejemplo se muestra el uso de propiedades compatibles con .NET Framework. Se pueden definir y establecer propiedades utilizando los métodos de descriptor de acceso de propiedad `get_` y `set_`.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo `Properties.vjsproj` y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo:

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **Properties.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar `vsvars.bat` (que se encuentra en el directorio `<%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools`). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio *<raíz de instalación de Microsoft Visual J++ 6.0>*.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo ValueTypes (utilizar tipos de valor de .NET Framework)

Este ejemplo muestra el uso de tipos de valor compatibles con .NET Framework. Visual J# no admite la creación (definición) de tipos de valor. Sin embargo, se pueden utilizar los tipos de valor definidos en bibliotecas de clases de .NET Framework.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo, abra el archivo ValueTypes.vjsproj y genere el ejemplo (Ctrl+Mayús+B).  
O bien
- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo:

- En el entorno de desarrollo, presione F5.  
O bien
- En la línea de comandos, escriba **ValueTypes.exe**.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)

# Ejemplo RemotingSample (Generar un servicio remoto)

Este ejemplo muestra cómo utilizar Visual J# para generar un servicio remoto. Este sencillo servicio manipula una cadena que se le ha pasado como parámetro y devuelve el resultado al llamador. Es importante tener en cuenta que el servicio debe estar alojado para que un cliente pueda llamarlo, lo que se puede lograr compilando el servicio como una DLL y registrándolo en IIS.

Este ejemplo contiene la siguiente estructura de directorios:

\bin

Contiene el archivo HelloService.dll necesario para tener acceso al servicio remoto mediante IIS. Hay que registrar el servicio en IIS para poder llamarlo desde los clientes de los directorios Clients\ConsoleClient y Clients\WebClient.

\Service

El código fuente para el servicio HelloService.dll y el archivo build.bat asociado se almacena aquí. Build.bat crea HelloService.dll y copia el archivo en el directorio bin.

\Client\ConsoleClient

Contiene el cliente y el archivo build.bat asociado. HelloService.dll se copia desde el directorio bin porque el compilador debe hacer referencia a los metadatos del servicio al compilar el cliente. Hay que registrar el servicio en IIS antes de ejecutar este cliente.

\Client\WebClient

Contiene los archivos .config y .aspx necesarios para tener acceso al servicio desde un explorador Web.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- Escriba **buildall.bat** desde la línea de comandos.

### Para registrar el servicio en IIS

1. Compile el servicio para generar un archivo DLL y copie este archivo DLL en el directorio bin desde el que se cargará el servicio. Para este ejemplo se crea el subdirectorio bin dentro del directorio del ejemplo y ahí se copia HelloService.dll. Para ello, genere el ejemplo ejecutando build.bat.
2. Inicie el Administrador de servicios Internet (accesible desde el menú **Inicio**, en **Herramientas administrativas**) y seleccione el sitio Web predeterminado en el nodo de servidor.
3. En el menú **Acción**, elija **Nuevo** y, a continuación, elija **Directorio virtual**.
4. Haga clic en **Siguiente** para continuar.
5. Escriba **HelloService** como alias y presione **Siguiente**.
6. Especifique el directorio en que se encuentra el servicio. Especifique la ruta completa, pero excluya el directorio bin en el que se copió el archivo dll en el paso 1; por ejemplo, C:\Samples\RemotingService.
7. Haga clic en **Siguiente**. Acepte los valores predeterminados y haga clic de nuevo en **Siguiente**. Haga clic en **Finalizar**.

El servicio remoto ya está registrado y aparecerá en la lista de sitios Web predeterminados en la ventana del Administrador de servicios Internet.

8. Compruebe que hay un archivo web.config en el directorio en que se registró el servicio. Este archivo describe el servicio y se cargará automáticamente cuando un cliente intente tener acceso al servicio con un explorador o un cliente que utilice el canal HTTP en el puerto 80.

### Para ejecutar este ejemplo

1. Compruebe que HelloService está registrado en IIS y ejecute el archivo ejecutable del directorio Clients\ConsoleClient.
2. Compruebe que el servicio HelloService está registrado en IIS y ejecute el ejemplo desde un explorador conectándose a <http://localhost/HelloService/Clients/WebClient>.

Es posible que deba reiniciar la máquina después de instalar Visual J# para que este ejemplo funcione correctamente.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas**, **Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).



Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio *<raíz de instalación de Microsoft Visual J++ 6.0>*.

## **Vea también**

[Ejemplos de tecnología](#)

# Ejemplo WFCUsingWinForm (ampliar una aplicación WFC generada en Visual J++ 6.0 con Windows Forms)

Este ejemplo muestra cómo extender una aplicación WFC generada en Visual J++ 6.0 con formularios Windows Forms. Se actualiza una aplicación WFC sencilla generada de Visual J++ 6.0 a Visual J#. Este ejemplo se extiende para iniciar un componente de formularios Windows Forms sencillo haciendo clic en un botón o seleccionando una opción de menú. Se puede utilizar un proceso similar para invocar e iniciar cualquier componente de formularios Windows Forms desde una aplicación WFC.

## Generar y ejecutar el ejemplo

### Para generar este ejemplo

- En el entorno de desarrollo integrado (IDE), abra WFCUsingWinForm.sln y genere el ejemplo (Ctrl+Mayús+B).

O bien

- En la línea de comandos, escriba **BUILD.bat**.

### Para ejecutar este ejemplo

- En el entorno de desarrollo, presione F5.

O bien

- En la línea de comandos, escriba **WFCUsingWinForm.exe**.

### Para utilizar la aplicación WFC

1. Haga clic en el botón para iniciar un componente de Windows Forms.
2. En el menú **File**, elija **Start Win Form** para iniciar el mismo componente de Windows Forms.

## Requisitos de la línea de comandos

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe ejecutar vsvars.bat (que se encuentra en el directorio <%Archivos de programa%>\Microsoft Visual Studio .NET 2003\Common7\Tools). Como alternativa, también puede utilizar la ventana de símbolo del sistema de Visual Studio .NET (haga clic en **Inicio**, elija **Programas, Microsoft Visual Studio .NET 2003** y, a continuación, **Visual Studio .NET Tools**).

Para que los ejemplos de Visual J# funcionen correctamente al generarlos y ejecutarlos desde la línea de comandos, debe configurar además la variable **Path** para que incluya el directorio <raíz de instalación de Microsoft Visual J++ 6.0>.

## Vea también

[Ejemplos de tecnología](#)