

TABLA DE CONTENIDO

Unidad TEMÁTICA N° 1	11
DISEÑO DE APLICACIONES DISTRIBUIDAS	11
1.1.- Desplazamiento de la arquitectura y desafíos del programador	11
Cambios en la arquitectura	11
Desplazamientos de la tecnología y desafíos del programador	11
Entornos heterogéneos	12
Escalabilidad	12
Rápido desarrollo de aplicaciones y de implementación	12
Administración de plataformas y gestión	13
Aplicaciones compatibles con redes	13
1.2.- Información General sobre aplicaciones distribuidas	13
Filosofía y ventajas	14
Servicios de presentación	14
Lógica empresarial/ Servicios de la aplicación.	14
Acceso a datos y servicios de almacenamiento	14
Servicios del sistema	15
1.3.- Objetivos del Diseño de Sistemas	15
Disponibilidad	15
Aumentar los niveles de disponibilidad	18
Aplicación no comercial	18
Aplicación comercial	19
Aplicación crítica para la empresa	19
Aplicación crítica para una misión	20
Facilidad de administración	20
Rendimiento	21
Confiabilidad	22
Utilizar herramientas de desarrollo de calidad	24
Pensar en la confiabilidad	24
Utilizar pruebas inteligentes	24
Escalabilidad	24
Escalar en vertical	25
Escalar en horizontal	26
1.4.- Introducción a la plataforma .NET	27
El entorno de ejecución CLR	28
1.5.- El Lenguaje Intermedio y el CLS	28
¿Cómo se consigue esta potente capacidad?	28
La especificación común de los lenguajes y el sistema de tipos comunes	29
1.6.- La biblioteca de clases de .NET	29
Los espacios de nombres	30
1.7.- Acceso a datos con ADO.NET	31
Arquitectura de ADO.NET	31
Capa conectada	32
Capa desconectada	32



1.8.- Aplicaciones Windows Forms	33
1.9.- Aplicaciones Web Forms	34
Unidad TEMÁTICA N° 2	39
CARACTERÍSTICAS DEL LENGUAJE C#	39
2.1.- El sistema de Tipos	39
Tipos de datos de .NET	39
Tipos primitivos	39
Sufijos o caracteres y símbolos identificadores para los tipos	40
Promociones numéricas implícitas (automáticas)	41
Conversiones numéricas explícitas	42
¿Cuándo debemos usar las conversiones explícitas?	42
¿Siempre funcionan las conversiones explícitas?	42
Funciones de conversión	43
Tipos por valor y tipos por referencia	43
2.2.- Variables y constantes	44
Declarar constantes	44
Declarar variables	44
Declarar variables y asignar el valor inicial	45
El tipo de datos char	45
Cadenas de caracteres	45
2.3.- Sentencias Condicionales y Repetitivas	46
Sentencia if-else	46
Sentencia switch	46
Sentencia for	47
Sentencia while	47
Sentencia do	48
2.4.- Arrays (matrices)	48
Declarar arrays	48
Declarar e inicializar un array	49
Cambiar el tamaño de un array	50
Eliminar el contenido de un array	50
Los arrays son tipos por referencia	51
2.5.- Manejo de excepciones	51
Manejo de excepciones estructuradas	51
Bloque try	52
Bloque catch	52
Varias capturas de errores en un mismo bloque try/catch	53
Bloque finally	53
Captura de errores no controlados	54
Estructuras: Tipos por valor definidos por el usuario	54
Definir una estructura	55
Constructores de las estructuras	55
Destructores de las estructuras	55
Los miembros de una estructura	56
Campos	56
Métodos y otros elementos	56
Cómo usar las estructuras	56

Unidad TEMÁTICA N° 3	59
DESARROLLO DE APLICACIONES WINDOWS	59
3.1.- Uso del diseñador de visual Studio 2005	59
Cuadro de herramientas	59
3.2.- Explorador de base de datos	61
Conexión a una base de datos Microsoft SQL Server 2005 a través de OLE DB	61
3.3.- Explorador de soluciones	64
3.4.- Propiedades	65
3.5.- Menús y barra de botones	66
3.6.- Otras consideraciones	67
3.7.- Controles Windows Forms	70
Datos	70
3.8.- Componentes	72
3.9.- Otros controles a tener en cuenta	73
Unidad TEMÁTICA N° 4	79
CREACIÓN DE CONTROLES	79
4.1.- Clases y controles personalizados	79
4.2.- Controles contenedores	86
4.3.- Generación de código rápido	89
Unidad TEMÁTICA N° 5	93
APLICACIONES CON ACCESO A DATOS	93
5.1.- Acceso a Datos	93
5.2.- ADO.NET	93
¿Qué es ADO.NET?	93
¿Qué capas o qué partes hay dentro de ADO.NET?	94
¿Qué nos permite realmente ADO.NET cuando trabajamos con XML?	94
5.3.- System.Data	95
La clase DataSet	95
La clase DataView	95
La clase DataTable	95
Un ejemplo práctico	95
5.4.- Los proveedores de acceso a datos	96
Proveedores de acceso a datos de .NET Framework	96
Otros proveedores de acceso a datos	97
El objeto Connection	97
El objeto Command	98
El objeto DataAdapter	98
El objeto DataReader	98
5.5.- El concepto DataBinding	98
El uso de DataBind	98
Unidad TEMÁTICA N° 6	103
ACCESO CONECTADO A BASE DE DATOS	103
6.1.- El paradigma de la conexión	103
6.2.- Conociendo el objeto DataReader	104
DataReader es de solo lectura	104
DataReader se maneja en una sola dirección	104



DataReader es rápido.....	104
Analizando el flujo de trabajo de DataReader	104
6.3.- Un primer contacto con el objeto DataReader	105
Un ejemplo simple para entenderlo mejor	105
6.4.- ¿Trabaja DataReader en un ambiente conectado realmente?	106
Desenchufando la fuente de datos usando DataReader	106
6.5.- Usando DataSource con DataReader	108
Demostración del uso de DataSource con DataReader	108
Carga segmentada de datos con DataSource y DataReader	110
6.6.- Usando los componentes de acceso a datos de .NET	111
Demostración del uso de BindingSource y BindingNavigator	111
Unidad TEMÁTICA N° 7	115
ACCESO DESCONECTADO A BASE DE DATOS DATASET Y DATAADAPTER	115
7.1.- Esquema general de la estructura desconectada de acceso a datos ...	115
Connection, DataAdapter y DataSet	115
7.2.- Conociendo el objeto DataAdapter	115
Connection, DataAdapter y DataSet	116
Utilizando las clases de .NET	116
Utilizando los componentes de .NET	118
7.3.- Insertando datos a través del objeto DataAdapter	124
¿Cómo se insertan datos con el objeto DataAdapter	125
Trabajando con un ejemplo	125
7.4.- Actualizando datos a través del objeto DataAdapter	128
7.5.- Eliminando datos a través del objeto DataAdapter	129
Unidad TEMÁTICA N° 8	133
ENLACE A FORMULARIOS CON DATASETS TIPADOS	133
8.1.- ¿Qué son los DataSets tipados?	133
Cómo trabajar con un DataSet tipado	133
¿Qué ventajas nos aportan los DataSets tipados?	133
8.2.- Generando nuestros DataSets tipados	134
Diagrama de datos	134
8.3.- Generando un DataSet tipado con Visual Studio 2005	135
Usando el entorno de desarrollo rápido Visual Studio 2005	135
8.4.- Usando los DataSets tipados	139
Usando las herramientas automáticas para trabajar con DataSets tipados	140
Usando DataAdapter con DataSets tipados	143
8.5.- ¿Qué son los datos Maestro Detalle?	154
8.6.- Configurando la fuente de datos	155
Configurando el origen de la fuente de datos	155
8.7.- Preparando el origen de datos	159
Preparando la tabla padre	159
Preparando la tabla hija	162
8.8.- Incrustando los datos maestro detalle	165
Incrustando la tabla padre en el formulario	165
Incrustando la tabla hija en el formulario	167
Relacionando la tabla padre con la tabla hija	169



8.9.- Manipulando los datos maestro detalle	172
Modificando datos.....	172
Insertando y eliminando datos	172

Unidad Temática I

DISEÑO DE APLICACIONES DISTRIBUIDAS

Dado que las aplicaciones distribuidas están convirtiéndose rápidamente en la plataforma preferida para la asistencia técnica de toda la empresa, la infraestructura necesaria para desarrollar y alojar las aplicaciones ha crecido tanto en escala como en complejidad. La plataforma .NET es el resultado del importante cambio registrado en la arquitectura de las aplicaciones informáticas durante la década de los noventa.

El presente curso tiene como objetivo el brindar los conceptos y herramientas necesarias para que usted pueda implementar una aplicación distribuida, teniendo en cuenta todos los parámetros y estándares necesarios para cumplir tal fin.

Para obtener un mejor aprendizaje el curso esta organizado en varios temas que van desde la descripción reconceptos relacionados con la fundamentos de programación y en particular el lenguaje C# , hasta la implementación de aplicaciones distribuidas utilizando un servidor de base de datos.

1.1.- Desplazamiento de la arquitectura y desafíos del programador

Para apreciar al máximo el significado de la plataforma .NET, hay que observar los cambios en su arquitectura, las causas de ello, y la continua repercusión en la comunidad informática.

Cambios en la arquitectura

Cuando la tecnología de Internet (especialmente la tecnología Web) se convirtió en la corriente dominante a mediados de los años noventa, el modelo aplicado a la informática de negocios cambió drásticamente. El cambio se centró en la noción que tenían las empresas del modelo informático cliente/servidor, que hasta entonces había sido muy complejo, costoso y exclusivo.

Ese modelo Web se puede caracterizar como un conjunto de niveles débilmente conectados formados por colecciones de aplicaciones e información que residen en una amplia gama de plataformas de hardware. No debe olvidarse que lo que mueve a Internet desde sus inicios es el deseo de crear una plataforma común de suministro de información, que sea escalable, extensible y de fácil acceso. Esta plataforma debe diseñarse para ser flexible y no puede verse limitada a uno o dos niveles de equipos. Los únicos límites para el desarrollo de aplicaciones de Internet son la capacidad de los equipos y la imaginación de los diseñadores.

Como los exploradores Web se hicieron rápidamente omnipresentes y los servidores Web proliferaron en todas las empresas, estaba claro que —a pesar del esfuerzo de los productores de software cliente/servidor para hacer compatibles sus productos con la tecnología Web— los programadores tuvieron que concebir de modo radicalmente diferente el modelo de aplicaciones. La primera idea que vino a la cabeza de los programadores para la informática de negocios fue la del "mínimo común denominador". Naturalmente, fueron necesarias nuevas técnicas y herramientas para hacer frente a los cambios y desafíos a los que se enfrentaban los programadores.

Desplazamientos de la tecnología y desafíos del programador

Cuando se consagró la revolución de Internet y aparecieron las nuevas tecnologías, los programadores se enfrentaron a multitud de retos que los modelos de diseño y las herramientas existentes en ese momento no podían gestionar adecuadamente. Las siguientes cuestiones eran primordiales en las decisiones de los programadores:

- Entornos heterogéneos
- Escalabilidad
- Rápido desarrollo de aplicaciones y de implementación
- Administración de plataformas y gestión
- Aplicaciones compatibles con redes

Entornos heterogéneos

Uno de los primeros, y quizás de los más importantes retos, fue el de generar aplicaciones que pudieran adaptarse fácilmente a entornos de trabajo heterogéneos. La mayoría de organizaciones tenía una gran diversidad de terminales, aplicaciones cliente complejas, y aplicaciones cliente para entorno Web sencillas. Además, para adaptarse al sistema de aplicaciones del tipo "cliente", los nuevos programas debían interactuar con datos heredados y con aplicaciones alojadas en grandes sistemas informáticos o en equipos convencionales, que procedían a menudo de diferentes proveedores de hardware.

Escalabilidad

Antes de la pujanza de las tecnologías Web, la escalabilidad era una cuestión relativamente fácil de administrar. Por entonces, el entorno de trabajo informático era fundamentalmente un sistema cerrado, porque el acceso remoto del personal, de los clientes o de los socios comerciales era limitado. Eso quería decir que el número de usuarios y sus costumbres de uso para aplicaciones y servicios concretos se conocían perfectamente. Los diseñadores de estrategia contaban con un amplio historial en el que basar sus proyecciones de escalabilidad del entorno informático, con objeto de adecuarse a las demandas del consumidor.

Además, por regla general, el desarrollo de una aplicación abarcaba varios años. Una vez más, los diseñadores contaban con mucho tiempo para idear la escalabilidad de sistemas y aplicaciones.

Por último, los equipos personales aún no habían alcanzado todo su potencial y las corporaciones acababan de empezar a instalar PCs en toda la empresa. De hecho, muchos consideraban los equipos personales como algo ligeramente más inteligente que un terminal. Con el paso del tiempo, se veía con expectación que los equipos personales formaran parte de cualquier aplicación.

Mientras el equipo personal redefinía la manera de trabajar de la gente, la tecnología de Internet (especialmente la tecnología Web) cambió las ideas de las empresas. En un principio, las empresas vieron esta nueva tecnología como un método ideal y económico para compartir información interna. Pero no sólo era barato, sino que, además, los usuarios podían llevar a cabo fácilmente sus propios desarrollos. Los sitios Web internos (intranets) empezaron a poblar rápidamente el paisaje informático.

Las proyecciones tradicionales de escalabilidad empezaron a tambalearse, y cuando las compañías abrieron sus puertas al mundo, se derrumbaron completamente. El nuevo modelo de diseño exigía plantear sistemas que se adaptaran de una base de usuarios de menos de un centenar a otra de más de un millón.

Rápido desarrollo de aplicaciones y de implementación

Los fenómenos intranet e Internet pusieron de manifiesto la posibilidad —y la necesidad— de la rápida implementación de aplicaciones. La experiencia de las intranets corporativas mostró claramente que era posible generar aplicaciones empresariales rápidamente. A esto venía a sumarse la sencillez de implementación URL. El resultado fue que los responsables empresariales y los usuarios empezaron a replantearse la totalidad de plataformas y procesos



de desarrollo tradicionales. Ya no estaban dispuestos a tener que esperar varios años antes de ser capaces de manejar una aplicación. Desde el punto de vista de la inversión, la comunidad empresarial empezó a poner en duda cualquier inversión en aplicaciones que terminara siendo un sistema heredado en el momento de completarse.

Las empresas ajustaron aún más la noción de "rápido desarrollo de aplicaciones" a partir del momento en que abrieron sus horizontes de aplicaciones de la intranet a Internet. Para ser competitivos, los programadores necesitaban crear aplicaciones prácticamente sobre la marcha para un uso inmediato: es lo que se denomina desarrollo just-in-time (JIT). Para conseguirlo, necesitaron renovar y revitalizar completamente su enfoque sobre el desarrollo de aplicaciones.

Administración de plataformas y gestión

Como ocurre en cualquier aspecto de la tecnología informática, las cosas no son perfectas en el mundo de Internet y la Web. Los profesionales de las tecnologías de la información (TI) que adoptaron este nuevo modelo de aplicaciones descubrieron que esa recién estrenada libertad y flexibilidad les aportaba un nuevo conjunto de características de administración y gestión. Esos aspectos estaban relacionados con clientes, aplicaciones y hosts.

El explorador dejó a la mayoría de las organizaciones sin un estándar. Incluso los problemas cotidianos de asistencia y actualización eran a menudo una pesadilla logística. Desde el punto de vista del desarrollo, la falta de estandarización obligaba a los diseñadores de aplicaciones a adaptar a cada versión del explorador las capacidades de procesamiento HTML (tanto básicas como avanzadas).

La implementación de aplicaciones era todavía más difícil de gestionar, en la medida en que los administradores de sistemas se enfrentaban más bien al creciente número de personas que publicaban contenidos que a un único grupo de programadores. La gestión de este aspecto de la informática basada en tecnología Web iba haciéndose cada vez más difícil a medida que las empresas aceptaban la idea de proporcionar contenido orientado a datos; o sea, dinámico. La necesidad de incluir diversos sistemas de almacenamiento de datos y de adaptar diferentes lenguajes de secuencias de comandos amplió aún más el ámbito del modelo de programación Web.

Cualquier administrador de un sitio Web de la época en que nacieron las aplicaciones Web para empresas podrá dar fe de la cantidad de horas de concienzudo trabajo manual que se requerían para mantener de forma correcta y continuada un sitio Web de mediano tamaño, porque otro aspecto del fenómeno Internet es que los usuarios deseaban acceso las 24 horas del día. Agregar servidores para adaptarse al creciente tráfico de los sitios Web supuso una mayor demanda de asistencia.

Desafortunadamente, los diseñadores de sitios Web y los abogados olvidaron incluir un conjunto de herramientas para gestionar la plataforma, y dejaron en manos de la comunidad de las Tecnologías de la Información la búsqueda de una solución.

Aplicaciones compatibles con redes

El último reto al que tenían que enfrentarse los programadores venía dado por los avances en la tecnología de equipos portátiles (PC, agendas electrónicas y palmtops) y su reducción en el precio. Gracias al acceso universal que proporciona Internet, la informática móvil ha crecido en una proporción comparable a la experimentada por la propia Red. Las cifras actuales indican que las ventas de equipos portátiles ya superan las de los equipos de escritorio.

Sea fuera de línea o bien desconectado, su uso ya no puede considerarse excepcional. La comunidad de usuarios espera poder utilizar aplicaciones y servicios tanto conectados a la Red como desconectados. El programador debe ser capaz de incorporar esta característica a las aplicaciones.

1.2.- Información General sobre aplicaciones distribuidas

Los arquitectos de aplicaciones pueden usar la plataforma .NET para desarrollar, implementar y dar asistencia a las aplicaciones distribuidas. Esta plataforma, altamente integrada, pero flexible, permite a los programadores generar soluciones empresariales completas que permiten aprovechar las arquitecturas y aplicaciones existentes.

Windows DNA era una arquitectura para generar aplicaciones distribuidas basadas en Web y de estricta correspondencia. Cuando las aplicaciones distribuidas empezaron a requerir unos principios de correspondencia menos precisos, la arquitectura de Microsoft cambió a la de la plataforma .NET.

Filosofía y ventajas

El principio fundamental de las aplicaciones distribuidas es la división lógica de una aplicación en tres capas fundamentales:

- Presentación
- Lógica empresarial
- Acceso a datos y almacenamiento

Los programadores pueden generar aplicaciones altamente escalables y flexibles, organizando las aplicaciones según estos criterios. Para ello, deberán usar técnicas de programación basadas en componentes y emplear al máximo las características de la plataforma .NET y del sistema operativo Microsoft Windows.

Un modelo simple de aplicación distribuida consiste en un cliente que comunica con la capa intermedia, que a su vez es el servidor de aplicaciones y una aplicación que contiene la lógica empresarial. La aplicación, a su vez, comunica con una base de datos que suministra y almacena datos.

Servicios de presentación

La capa de presentación incluye tanto una interfaz de cliente enriquecida (o sencilla) como una aplicación. El cliente enriquecido proporciona al sistema operativo — ya sea directamente mediante la API Win32 de Microsoft o indirectamente mediante formularios Windows Forms— una interfaz completa de programación, y usa ampliamente sus componentes.

El cliente sencillo (explorador Web) se está convirtiendo rápidamente en la interfaz preferida de muchos programadores. Un programador puede generar lógica empresarial que pueda ejecutarse en cualquiera de los tres niveles de la aplicación. Con las aplicaciones para Web ASP.NET y los servicios Web XML, el cliente sencillo puede proporcionar a las aplicaciones una interfaz de usuario visualmente rica, flexible e interactiva. Los clientes sencillos también tienen la ventaja de aportar un mayor grado de portabilidad entre plataformas.

Lógica empresarial/ Servicios de la aplicación.

Esta capa se divide en servidores de aplicaciones y servicios, disponibles para ayudar a los clientes. Las aplicaciones Web pueden escribirse para sacar partido de los servicios COM+, de Message Queuing (MSMQ), de los servicios de directorio y de los servicios de seguridad mediante .NET Framework. Los servicios de la aplicación, por el contrario, pueden interactuar con multitud de servicios en la capa de acceso a datos.

Acceso a datos y servicios de almacenamiento

Los servicios de datos que admiten acceso a datos y almacenamiento son:

- ADO.NET, que proporciona un acceso simplificado a los datos mediante programación, ya sea por medio de lenguajes de secuencias de comandos o de programación.
- OLE DB, que es un proveedor de datos universal desarrollado por Microsoft.
- XML, que es un formato estándar para especificar estructuras de datos.

XML es un estándar escogido por la comunidad internauta. Mientras que HTML se centra en el modo en el que el explorador procesa la información y la presenta en pantalla, el objetivo de XML es manejar la estructura de datos y su representación.

Servicios del sistema

.NET Framework y el sistema operativo Windows aceptan en su totalidad los elementos que conforman cada segmento de este modelo. Entre sus muchos servicios se encuentran los de directorio, seguridad, administración y comunicaciones, que operan en las tres capas. Las herramientas de programación, que incluyen el sistema de desarrollo Visual Studio .NET, permiten a los programadores generar componentes de aplicaciones entre las diferentes capas.

1.3.- Objetivos del Diseño de Sistemas

Los objetivos del diseño de una aplicación se establecen durante la fase de diseño del desarrollo de la misma y se describen a continuación.

Disponibilidad

Todas las aplicaciones están disponibles al menos durante parte del día, pero las aplicaciones basadas en Web y las aplicaciones empresariales críticas para una misión deben proporcionar, por lo general, servicios de veinticuatro horas. Si una determinada aplicación empresarial tiene que funcionar las 24 horas del día, los 7 días de la semana, es muy probable que tenga que diseñarla para que ofrezca una alta disponibilidad. Los avances que se han producido tanto en hardware como en software han aumentado de manera espectacular la calidad de las aplicaciones de alta disponibilidad. Sin embargo, la disponibilidad no es fácil de implementar y requiere una infraestructura de diseño considerablemente más compleja que la generación previa de aplicaciones cliente/servidor.

Si la aplicación requiere una alta disponibilidad, es conveniente que comprenda la forma en que las opciones de diseño ayudan a maximizar la disponibilidad de la aplicación, y la forma en que las pruebas realizadas pueden validar los niveles de servicio previstos.

Las aplicaciones basadas en Web de hoy en día normalmente dependen de varios servidores, de cientos o tal vez miles de estaciones de trabajo de cliente, de comunicaciones a través de redes internas y externas, de servicios de base de datos, de procesos operativos y de una gran cantidad de otros servicios de infraestructura que deben funcionar conjuntamente y de manera uniforme. Allí donde el ideal de negocio es un flujo continuo de información, y donde una interrupción supone gastos para la compañía, la creación de aplicaciones de alta disponibilidad se convierte en una estrategia comercial importante.

Las compañías que confían cada vez más en aplicaciones distribuidas basadas en Web para llevar a cabo actividades comerciales importantes necesitan una gran variedad de opciones de diseño de disponibilidad, para satisfacer los requisitos de nivel de servicio de forma rentable. En la práctica, no es necesario que todas las aplicaciones funcionen ininterrumpidamente y proporcionen una respuesta instantánea. Es posible que algunas aplicaciones produzcan errores sin consecuencia alguna. Otras aplicaciones pueden tolerar tiempos de inactividad imprevistos, pero requieren diferentes estrategias de recuperación; y existen aplicaciones que deben proporcionar una disponibilidad muy alta a través de estrategias de replicación en suspenso, que han de garantizar una recuperación instantánea y transparente sin apenas tiempos de inactividad perceptibles.

A medida que aumentan la complejidad y los niveles de acceso de las aplicaciones distribuidas, también aumenta la probabilidad de que surjan errores en el diseño original, en los servicios de soporte técnico, o en el mantenimiento y las mejoras realizadas desde que las aplicaciones se instalaron por primera vez. Lamentablemente, los tiempos de inactividad originan problemas y

pérdidas a largo plazo que van más allá del propio entorno informático local (como el enojo de los clientes y la ruptura de los canales de suministro).

Tal y como se describe en el tema referente a la confiabilidad, se pueden producir errores en las aplicaciones por muchos motivos:

- Comprobación inadecuada
- Problemas relacionados con cambios en la administración
- Falta de control y análisis continuados
- Errores en las operaciones
- Código poco consistente
- Ausencia de procesos de diseño de software de calidad
- Interacción con aplicaciones o servicios externos
- Condiciones de funcionamiento distintas (cambios en el nivel de uso, sobrecargas máximas)
- Sucesos inusuales (errores de seguridad, desbordamientos en la difusión)
- Errores de hardware (discos, controladores, dispositivos de red, servidores, fuentes de alimentación, memoria, CPU).
- Problemas de entorno (red eléctrica, refrigeración, incendios, inundaciones, polvo, catástrofes naturales)

Alrededor de un 40 por ciento del tiempo de inactividad de las aplicaciones se debe a una comprobación inadecuada, a una administración de cambios y a la ausencia de un control continuo de los errores. Otro 40 por ciento se debe a errores operativos producidos por la ausencia de procedimientos rigurosos y a errores de copia de seguridad/restauración. La confiabilidad del hardware ha mejorado durante los últimos años que menos del 10 por ciento del tiempo de inactividad se debe a problemas de hardware. El 10 por ciento restante del tiempo de inactividad se debe a problemas de entorno y a problemas de otro tipo.

En términos generales, se puede decir que la disponibilidad es una medida de la frecuencia con la que se puede utilizar la aplicación. Para ser más exactos, la disponibilidad es un cálculo porcentual del tiempo en que la aplicación está realmente disponible para controlar las solicitudes de servicio en comparación con el tiempo de ejecución total disponible previsto. El cálculo formal de la disponibilidad incluye el tiempo de reparación, ya que una aplicación que se está reparando no está disponible.

En el cálculo de la disponibilidad se utilizan varias medidas:

Nombre	Acrónimo	Cálculo	Definición
Tiempo medio entre errores	MTBF	Horas / número de errores	Duración media de funcionamiento de la aplicación antes de que produzca errores.
Tiempo medio de recuperación	MTTR	Horas de reparación / número de errores	Tiempo medio necesario para reparar y restaurar el servicio después de que se produzca un error.

La fórmula de disponibilidad tiene esta forma:

$$\text{Disponibilidad} = (\text{MTBF} / (\text{MTBF} + \text{MTTR})) \times 100$$



Considere, por ejemplo, una aplicación concebida para que funcione continuamente. Pongamos un punto de control de 1 000 horas consecutivas, dos errores de una hora durante ese período darían lugar a una disponibilidad de $((1\ 000/2) / ((1\ 000/2) + 1)) \times 100 = (500 / 501) \times 100 = 0,998 \times 100 = 99,8 \%$.

Una forma conocida de describir la disponibilidad es mediante los "nueves": los tres nueves de una disponibilidad de un 99,9%. No obstante, hay que tener en cuenta que las implicaciones de la medición por nueves a veces se malinterpretan. Es necesario realizar algunos cálculos para descubrir que tres nueves (disponibilidad de un 99,9%) representan aproximadamente 8,5 horas de interrupción de servicio en un solo año.

El nivel inmediatamente superior, cuatro nueves (99,99%), representa alrededor de una hora de interrupción de servicio en un año. Cinco nueves (99,999%) representan sólo cinco minutos de interrupción al año.

Conforme se alcanzan niveles más altos de disponibilidad, son varias las cosas que suceden en el proyecto:

- En primer lugar, aumentan los costes de hardware de la aplicación debido a la redundancia de servidores, redes y discos.
- En segundo lugar, la identificación y eliminación de errores complejos se convierte en una tarea cada vez más difícil, que requiere ingenieros de software muy cualificados y especializados.
- En tercer lugar, una alta disponibilidad requiere una comprobación exhaustiva de cada uno de los procesos automáticos y manuales que pueden afectar a la aplicación mientras ésta se encuentra en servicio.

Las buenas noticias son que la mayoría de las aplicaciones empresariales pueden funcionar eficazmente con una disponibilidad de un 99,9%. Si se dispone de la combinación adecuada de personas, procesos de diseño y tecnología, la consecución de los "tres nueves" se convierte en una tarea muy accesible, asequible y común en lo que se refiere a acuerdos habituales relacionados con el nivel de servicio.

Al igual que ocurre con la confiabilidad, la selección de una infraestructura tecnológica de software y hardware adecuada ayuda ciertamente a obtener una alta disponibilidad; no cabe duda de que determinadas opciones de diseño tecnológico son cruciales a la hora de crear aplicaciones de alta disponibilidad.

Sin embargo, no es posible alcanzar altos niveles de disponibilidad sin un compromiso serio de adquisición de personal cualificado, procesos de calidad durante el ciclo vital y calidad funcional. Es difícil darse cuenta de que las aplicaciones de alta disponibilidad deben generalmente su éxito a la forma en que se acometen, y no a una combinación particular de tecnologías.

Realizar planes referentes a niveles de disponibilidad

No es fácil determinar el nivel de disponibilidad que satisfará unos requisitos empresariales determinados.

En primer lugar, es difícil calcular cuál será la disponibilidad real necesaria para satisfacer el nivel de servicio comercial previsto, así como las previsiones presupuestarias y del calendario. Además, el modelo de uso y el entorno de ejecución del software pueden variar con el tiempo. Esto último distorsiona las previsiones de disponibilidad originales y puede requerir una reconsideración del acuerdo de servicio de disponibilidad original.

Antes de decidir el nivel de disponibilidad adecuado para la aplicación, es necesario responder a unas cuantas preguntas:

- ¿Quiénes son los clientes y cuáles son sus expectativas?

- ¿Cuál es el tiempo de inactividad aceptable?
- ¿Los procesos internos de la compañía dependen del servicio?
- ¿Cuál es el calendario y con qué presupuesto se cuenta?

Si la aplicación en cuestión es un sitio Web público, será difícil prever la carga de trabajo del cliente. Deberá responder a algunas cuestiones básicas como el número de clientes previsto y el momento del día en el que utilizarán el sitio Web. Previsiblemente, el documento de estrategia empresarial en el que se basó la creación del sitio Web contendrá una descripción del perfil del cliente.

Los calendarios y el presupuesto de un proyecto no deben entrar en conflicto con el requisito de disponibilidad crítica para una misión o para la empresa. Aunque sea muy tentador crear e implementar una versión más rudimentaria de la aplicación, no lo haga: el coste de los problemas de mantenimiento ininterrumpido y la pérdida de clientes no merecen la pena. Si debe implementar una aplicación de alta disponibilidad y el presupuesto y los calendarios no son viables, haga lo correcto: redirija el alcance del proyecto y consiga los fondos adecuados.

El diseño de la disponibilidad presenta desafíos importantes. Dada la gran variedad de arquitecturas de aplicación, no existe una sola solución de disponibilidad que funcione en todas las situaciones. La decisión de implementar una solución exhaustiva, tolerante a errores, totalmente redundante y con equilibrio de carga puede ser apropiada para el procesamiento de transacciones en línea de gran volumen. Por otro lado, algunas aplicaciones pueden aceptar tiempos de inactividad moderados con mínimas consecuencias para los clientes. En última instancia, las decisiones de diseño vienen condicionadas por una combinación de requisitos comerciales, datos específicos de la aplicación y el presupuesto disponible.

Entonces, ¿qué porcentaje de disponibilidad será el adecuado para una aplicación determinada? A continuación se incluyen algunas pautas:

Categoría	Número de errores anual	Tiempo de inactividad anual	Tiempo medio de reparación	Disponibilidad
No comercial	10	88 horas	10 horas	99.00%
Comercial	5	44 horas	8,8 horas	99.50%
Crítica para la empresa	4	8,5 horas	2,25 horas	99.90%
Crítica para una misión	4	1 hora	0,25 horas	99.99%

Nota Para realizar estos cálculos de disponibilidad se han utilizado $365,25 \times 24 = 8.766$ horas al año.

Aumentar los niveles de disponibilidad

Para comprender el verdadero significado de estos números, analicemos la forma en que se pueden utilizar las técnicas de disponibilidad para modernizar una aplicación existente. Considere una situación en la que se va a migrar una aplicación de su categoría inicial de aplicación no comercial a la categoría de aplicación crítica para una misión.

Aplicación no comercial

Como punto de partida, dispone de una aplicación no comercial que tiene 10 errores al año, con un tiempo de inactividad total de 88 horas al año. Se trata de una aplicación bastante común, con una disponibilidad de un 99,00%, lo que significa que se ejecuta la mayor parte

del tiempo y que, cuando se producen errores, se requiere la intervención de personal cualificado (y tal vez algunos programadores) para identificar el problema, concebir una solución, restaurar los datos y reiniciar la aplicación.

Aplicación comercial

Una nueva estrategia comercial exige mejorar la aplicación y adaptarla a los estándares comerciales. Si se desea mejorar la disponibilidad de esta aplicación de un 99.00 a un 99.50%, en primer lugar, deberá aplicarse cierto análisis a su arquitectura, así como rediseñar sus componentes para reducir el recuento de errores a 5. Al estar causados algunos de los problemas por un mal control de los mensajes de error, se deberá crear un proceso de control común y resolver varias condiciones de errores recuperables. Una mejora del diseño de los componentes puede requerir un proceso adicional de aprendizaje para el personal, especialmente en lo referente a la comprensión del diseño de software de confiabilidad y disponibilidad.

Aunque no sea decisivo, puede examinar también la infraestructura de apoyo que utiliza la aplicación (como la seguridad, los servicios Web, el acceso a los datos, las transacciones y las colas) y plantearse una sustitución de la tecnología antigua. Aunque haya reducido el porcentaje de errores a cinco por año, sigue siendo necesario reducir el tiempo medio de reparación de 10 a 8,8 horas. Imagine que decide reducir el tiempo de reparación creando un documento de recuperación de operaciones y ofreciendo información sobre el manejo de errores. Asimismo, como algunos de estos errores se produjeron debido a un control de configuración erróneo, implementa un sistema de control de cambios automatizado.

Dado que esta aplicación es cada vez más importante para la empresa, sospecha que necesitará alguna información que le ayude a analizar la aplicación para que pueda satisfacer los requisitos de disponibilidad futuros. Implementa la Instrumentación de Administración de Windows (WMI, Windows Management Instrumentation) y establece una consola de administración para ver datos de análisis de errores.

Sorprendentemente, la aplicación sigue guardando gran similitud con su forma original. El hardware es el mismo; ha mejorado algunos componentes, ha actualizado parte de la infraestructura y ha aplicado algunos procedimientos operativos nuevos para potenciar la coherencia y reducir el número de errores humanos.

Aplicación crítica para la empresa

Realizar la transición de una disponibilidad comercial (99,50%) a una disponibilidad crítica para la empresa (99,90%) es mucho más difícil. Supongamos que con la realización de pruebas intensivas y un rediseño adicional de los componentes consigue reducir los errores a sólo cuatro al año. Pero el tiempo de inactividad debe reducirse de 44 horas anuales a sólo 8,5 horas. Esto supone nada más y nada menos que una reducción del 80%. Evidentemente, es ahora cuando el diseño de una disponibilidad de solidez industrial se convierte en crucial para el proyecto.

Este paso requiere un compromiso total con una referencia cultural de confiabilidad y disponibilidad: cursos para el personal, procesos rigurosos de diseño de software de calidad, certificación de Windows Server y las tecnologías adecuadas.

Para empezar, puede tomar la decisión de que toda la lógica de la aplicación crítica esté controlada por transacciones que garanticen una finalización correcta y eviten que los datos se dañen. Además, puede hacer una revisión exhaustiva de la seguridad y centrarse en la protección de los datos y de la infraestructura de la aplicación para evitar posibles ataques o robos.

Puede también implementar Windows Advanced Server en cualquiera de sus versiones para proteger la memoria, los archivos y la instalación, proporcionar un mantenimiento de reinicio, un equilibrio de la carga en la red (NLB), un equilibrio de carga de los componentes (CLB) y amplios servicios de componentes distribuidos. Partiendo de que se alcanza una mayor

seguridad cuanto mayor sea el número de componentes, puede instalar discos RAID compartidos para almacenar los datos.

Ahora sí que sería crucial reducir el tiempo de recuperación, por lo que una buena estrategia inicial sería utilizar dos grupos (cluster) Web de interfaz de usuario, cada uno de ellos con varios servidores y configurados como grupos de conmutación por error mediante Microsoft Cluster Services. Esto último requeriría mejorar el código de la aplicación para restablecer las conexiones de base de datos y reiniciar las transacciones. Asimismo, puede agregar una lógica de "reintentar en caso de error" en la aplicación de cliente, y proporcionar una degradación si determinados recursos o servicios no están disponibles. Microsoft Cluster Services intentará primero reiniciar la aplicación en el mismo servidor. Si no es posible, moverá con rapidez los recursos de la aplicación y los reiniciará en otro servidor.

Por último, puede ampliar la capacidad de la consola de administración instalando Application Center para supervisar y controlar las cargas de trabajo y los cambios de configuración. La aplicación dispone ahora de una capacidad de recuperación de conmutación por error y de equilibrio de carga, y ya puede supervisar y ajustar de manera continuada la carga de trabajo entrante.

Aplicación crítica para una misión

La transición a una disponibilidad crítica total para una misión (99,99%) significa que la aplicación debe realizar sus servicios con sólo una hora de tiempo de inactividad anual. Teniendo en cuenta que la tasa de errores sigue siendo de cuatro errores anuales, el tiempo medio de reparación de cada error debe reducirse a sólo 15 minutos (0,25 horas). La consecución de una disponibilidad del 99,99% no es una cuestión trivial.

En estos momentos, empezará a darse cuenta de que las aplicaciones distribuidas basadas en Web comparten un esquema de diseño recurrente en el servidor de seguridad: servicios Web de cliente, servicios de datos de servidor e infraestructura de administración. La técnica principal para aumentar la disponibilidad es la redundancia, así que, si examina detenidamente la arquitectura de la aplicación, puede tomar la decisión de implementar una redundancia completa, que incluye servidores de aplicaciones para el usuario duplicados, infraestructura de red redundante y servidores de servicios de fondo duplicados.

Puede crear servidores de aplicaciones para usuarios, adaptarlos al equilibrio de la carga en la red y asegurarse de que todos los servicios de cliente sean independientes. Puesto que la conexión continua a Internet es ahora crucial, puede instalar varias conexiones mediante varios ISP a través de rutas de acceso a redes físicamente separadas. En el caso de los servidores de servicios de fondo, puede volver a utilizar Microsoft Cluster Services y proporcionar redundancia en el ámbito de los datos y de los servicios de conmutación por error. Este agrupamiento permite que varias bases de datos SQL y recursos compartidos de archivos residan en un dispositivo RAID de forma que, si un servidor produce un error, un servidor de reserva entra automáticamente en funcionamiento y continúa el proceso que ha producido errores. Por último y para finalizar la actualización, puede instalar varias fuentes de alimentación de reserva, sistemas de prevención de incendios y otros sistemas de protección del entorno.

Facilidad de administración

Cuando se encuentra ante aplicaciones distribuidas, los aspectos que más le preocupan son el lugar en el que se encuentran las cosas en la red, cómo se implementan y se configuran componentes y cómo se establecen los atributos de seguridad para 10.000 usuarios distintos. En algunas ocasiones puede llegar a preguntarse si la aplicación funciona de la forma prevista. Sin duda, no hay tiempo para desplazarse desde la oficina hasta las decenas (o quizá centenas) de ubicaciones remotas que requieren reparaciones o actualizaciones de software.

La administración de una aplicación empresarial siempre ha supuesto una parte importante del coste total de la propiedad. Al igual que ocurre con los precios del hardware y del



software, le gustaría que el precio de la administración continuada de la aplicación también disminuyese. Lo que necesita es una forma eficaz de implementar, configurar, actualizar y supervisar todos los componentes y servicios locales y remotos de la aplicación distribuida.

Si no tiene experiencia con aplicaciones empresariales, es posible que se pregunte cómo puede incluir funciones de administración en la aplicación. La respuesta la encontrará en Windows Server, que proporciona servicios de administración integrados, una infraestructura extensible y un conjunto de herramientas diseñadas para administrar aplicaciones de software.

En las siguientes secciones encontrará información sobre cómo diseñar e implementar una aplicación empresarial fácil de administrar.

Las aplicaciones distribuidas basadas en Web plantean un problema interesante. Mientras que el acceso de clientes y el intercambio de datos con socios comerciales es cada vez más sencillo, la capacidad de diagnosticar y resolver nuevos problemas de capacidad y recursos resulta cada vez más difícil. Se necesita algo más que llamadas a la API o entradas directas del Registro para realizar actividades comunes de administración de aplicaciones. Además, es necesario realizar auditorías de control de calidad en tiempo real que obliguen a incorporar las conclusiones del análisis en las actualizaciones de mantenimiento. La dificultad de estos procesos se debe a la necesidad de realizar todas estas actividades en una infraestructura de aplicación que suele estar distribuida a lo largo de una gran área geográfica.

La administración de una aplicación .NET moderna requiere un modo eficaz de controlar procesos normales de compatibilidad con la aplicación, tanto locales como remotos, entre los que se incluyen:

- Implementación inicial
- Ajuste de la configuración
- Mantenimiento programado y no programado
- Solución de problemas ocasionales

Además de estas cuestiones de funcionamiento típicas, la administración de aplicaciones, como necesidad empresarial, puede beneficiarse de la medición continua del estado de la aplicación, en la que se incluyen diversos factores como el rendimiento, las tendencias de consumo de recursos, las cargas de tráfico de clientes y la aparición de problemas.

Microsoft proporciona un conjunto de funciones eficaces incorporadas al sistema operativo Microsoft Windows Server que sirven para generar información de administración en la aplicación y para realizar actividades de administración periódicas.

Rendimiento

Las medidas clave de una aplicación, como el volumen de transacciones y el uso de recursos, definen el rendimiento de una aplicación. Las medidas relacionadas con el hardware, como el rendimiento de la red y el acceso a discos, son cuellos de botella comunes en el rendimiento de una aplicación. Desde el punto de vista de los usuarios, el tiempo de respuesta de una aplicación define el rendimiento. Por supuesto, el rendimiento no está libre de un precio. Si bien es posible crear una aplicación de alto rendimiento para cualquier espacio de problema dado, un punto de precio clave es el coste por transacción. A veces es necesario sacrificar el rendimiento para controlar el coste.

Rendimiento y escalabilidad no son términos equivalentes, pero es comprensible que se confundan. Los problemas de rendimiento no suelen ser transparentes hasta que las personas encargadas de probar las aplicaciones aumentan la carga de una aplicación. Sin embargo, el rendimiento y la escalabilidad son dos cuestiones diferentes. La medida del rendimiento de una aplicación cuando se aumenta cada vez más la carga determina la escalabilidad de esa aplicación. Cuando el rendimiento comienza a aminorar por debajo de

los requisitos mínimos de rendimiento ofrecidos, ha llegado al límite de la escalabilidad de la aplicación.

Descuidar el rendimiento de una aplicación lleva inevitablemente a un mal funcionamiento. El rendimiento de una aplicación queda determinado tanto en el tiempo de diseño como en el de ejecución.

En tiempo de diseño, los desarrolladores deben evitar la introducción de código que pudiera mermar el rendimiento de la aplicación. Los desarrolladores pueden contribuir siguiendo las prácticas de programación aceptadas y aprovechando las ventajas que ofrece la capacidad de mejora del rendimiento inherente del lenguaje de programación, el entorno de destino en tiempo de ejecución y los métodos de acceso a datos. Un método común para identificar código problemático es realizar revisiones del código de rutinas. Los obstáculos del rendimiento que se detectan y corrigen en este momento suelen ser más fáciles y baratos de reparar.

En tiempo de ejecución, la aplicación debe pasar pruebas de rendimiento obligatorias para identificar cuellos de botella, como la contención de recursos o código de ejecución lenta. Antes de someter una aplicación a pruebas exhaustivas de rendimiento, es crucial que se hayan completado todas las pruebas funcionales. Dicho de otro modo, para que una aplicación funcione bien antes es preciso conseguir que funcione. No obstante, las pruebas comparativas del rendimiento deben comenzar tan pronto como sea posible para identificar las áreas problemáticas al incluirlas en la aplicación. Contraste las pruebas comparativas sucesivas con la prueba comparativa inicial del rendimiento de la aplicación para determinar el progreso o la regresión. Realice estas pruebas sin variaciones innecesarias, como cambios de hardware, para permitir comparaciones precisas de las pruebas sucesivas. Una vez más, los obstáculos del rendimiento que se detectan en una etapa temprana resultan más fáciles y baratos de subsanar.

Supervise el rendimiento de la aplicación (sobre todo la productividad y la latencia) y ajuste los parámetros (software) y la configuración (hardware), para eliminar o minimizar cuellos de botella. Utilice el siguiente método:

- Determinar el problema.
- Crear una solución.
- Poner en práctica la solución.
- Analizar los resultados.
- Documentar todo.

Confiabilidad

A medida que aumenta el tamaño y la complejidad de las aplicaciones distribuidas, surge una necesidad cada vez mayor de mejorar la confiabilidad y la calidad de funcionamiento del software. En primer lugar, el coste de los errores de la aplicación suele ser demasiado elevado. Los usuarios evitan los sitios Web poco confiables, lo que da lugar a una pérdida de ingresos y a una reducción de las futuras ventas; además, los gastos de reparación de los datos dañados pueden incrementar el coste de los errores de la aplicación. En segundo lugar, los sistemas poco confiables son difíciles de mantener y de mejorar, ya que el origen de los errores por lo general está oculto en el sistema. Por último, la moderna tecnología de software facilita la creación de aplicaciones confiables.

Deberá aprender a incluir características de confiabilidad importantes en el diseño de la aplicación y conseguir que la confiabilidad forme parte del ciclo vital de desarrollo del software.

En las secciones que se describen a continuación obtendrá información sobre las estrategias de diseño, comprobación y prácticas adecuadas para la creación de aplicaciones empresariales confiables.



Como las aplicaciones basadas en Web de hoy en día ejercen una gran influencia, que va desde la experiencia del cliente hasta las relaciones con los proveedores, los sistemas de información corporativos confiables son cada vez más importantes para la compañía, los clientes y los socios comerciales. Puesto que los errores de una aplicación pueden dar lugar a pérdidas de negocio y costes de recuperación importantes, y puesto que Internet está siempre abierto a los negocios, las compañías exigen una confiabilidad ininterrumpida para la mayoría de las aplicaciones empresariales. Internet ha convertido la información (y el acceso inmediato y sin errores a la misma) en el activo más valioso de una compañía.

Una aplicación empresarial es un conjunto de hardware, servicios del sistema operativo, componentes de software y (normalmente) procesos humanos, cuyo objetivo es proporcionar, conjuntamente, los servicios comerciales previstos. La confiabilidad de toda la aplicación depende en gran medida de la confiabilidad de los distintos componentes. Como todos los componentes del sistema están relacionados, un error en uno de ellos puede afectar a la confiabilidad del resto de los componentes.

Se producen errores en la aplicación por distintos motivos:

- Comprobación inadecuada
- Problemas relacionados con cambios en la administración
- Falta de control y análisis continuados
- Errores en las operaciones
- Código poco consistente
- Ausencia de procesos de diseño de software de calidad
- Interacción con aplicaciones o servicios externos
- Condiciones de funcionamiento distintas (cambios en el nivel de uso, sobrecargas máximas)
- Sucesos inusuales (errores de seguridad, desbordamientos en la difusión)
- Errores de hardware (discos, controladores, dispositivos de red, servidores, fuentes de alimentación, memoria, CPU).
- Problemas de entorno (red eléctrica, refrigeración, incendios, inundaciones, polvo, catástrofes naturales)

En términos generales, la confiabilidad del software se refiere a la precisión con la que una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales. El diseño para favorecer la confiabilidad, además de referirse al tiempo de funcionamiento de la aplicación antes de que se produzca algún error, está relacionado también con la consecución de resultados correctos y con el control de la detección de errores y de la recuperación para evitar que se produzcan errores.

En términos más formales, la confiabilidad se define como el tiempo medio entre errores (MTBF), es decir, el tiempo medio de funcionamiento de la aplicación antes de que se produzca algún error. La fórmula del MTBF es bastante simple:

$$\text{MTBF} = \text{Horas} / \text{número de errores}$$

Suponga, por ejemplo, que su equipo de diseño espera que la aplicación proporcione una confiabilidad de un error cada 30 días, si funciona las 24 horas del día (aproximadamente un error cada 720 horas). Las pruebas pueden mostrar que la aplicación finalizada funciona durante 1.800 horas con dos errores, por lo que el MTBF será de $1800/2 = 900$ horas. Dicho de otro modo, es bastante probable que la aplicación funcione unas 900 horas sin que se produzca ningún error. En este ejemplo, la aplicación tiene una confiabilidad que supera los parámetros de diseño.

Una noción importante sobre la confiabilidad del software es que los errores de software se producen mientras el software está disponible y se está ejecutando.

Como consecuencia de los errores pueden ocurrir varias cosas, desde que se interrumpa el servicio solicitado simplemente, hasta que se suministre un servicio incorrecto, que se produzcan datos erróneos y estados de sistema dañados. Mientras que algunos errores simplemente son incómodos (un sitio Web no responde, por ejemplo), otros errores pueden ir desde una interrupción que produce una pérdida de tiempo (el sistema de compras no rellena un pedido, por ejemplo), hasta problemas graves (un sitio de mercado bursátil está demasiado ocupado para reconocer su conexión, por ejemplo) e incluso problemas catastróficos (el sistema de navegación de una embarcación produce un error, por ejemplo).

Al considerar los diversos tipos de errores de la aplicación, puede parecer en un principio que la solución reside únicamente en la tecnología del hardware y del software. Aunque, en efecto, la tecnología es una de las soluciones, la mejora más grande en confiabilidad no consiste en conectar la infraestructura tecnológica, sino en disponer de personal bien preparado, procesos de diseño de software de calidad y un compromiso serio y continuo de confiabilidad.

Utilizar herramientas de desarrollo de calidad

Las herramientas de desarrollo deben permitir el desarrollo de aplicaciones tolerantes a errores y proporcionar un entorno fácil de utilizar para crear, comprobar e implementar aplicaciones. Microsoft proporciona Visual Studio .NET para el desarrollo rápido de aplicaciones distribuidas basadas en Web. Con Visual Studio .NET, puede generar aplicaciones que creen y utilicen los nuevos servicios Web mediante código administrado generado sobre Common Language Runtime.

Pensar en la confiabilidad

La creación de software de aplicaciones confiables requiere una observación del modo en que se utiliza la aplicación y una reflexión sobre los orígenes de errores de procedimiento. Las aplicaciones confiables han de ser compatibles con operaciones confiables y necesitan también procesos de implementación confiables. Céntrese en el modo en que se proporciona el servicio y busque posibles problemas allí donde las alternativas de diseño o de procedimiento permitan reducir las causas de error.

Utilizar pruebas inteligentes

Los procesos de prueba de control de calidad deberán proporcionar una respuesta a tres cuestiones importantes:

- ¿Están correctamente implementadas en la aplicación las funciones descritas en las especificaciones?
- ¿Satisface la aplicación las situaciones de usuario previstas sin producir errores?
- ¿Se ajusta el perfil de confiabilidad de la aplicación a los requisitos originales o los supera?

Cuando el nivel de calidad y confiabilidad no sea aceptable, deberá corregirse el software hasta que se alcance el nivel deseado.

Escalabilidad

La escalabilidad es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de servicio. La característica clave de una aplicación es que la carga adicional sólo requiere recursos adicionales en lugar de una modificación extensiva de la aplicación en sí.

Aunque el rendimiento marca una diferencia a la hora de determinar el número de usuarios que puede admitir una aplicación, la escalabilidad y el rendimiento son dos entidades

diferentes. De hecho, las labores de rendimiento pueden ser opuestas a veces a las de escalabilidad.

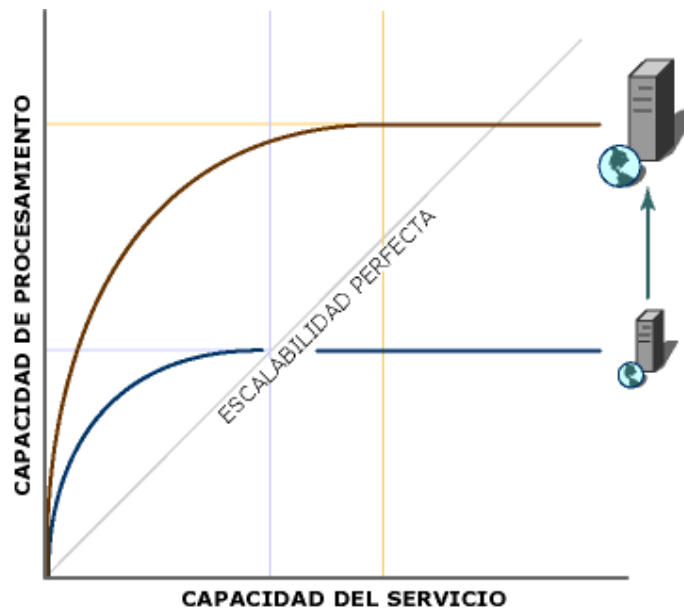
La escalabilidad debe formar parte del proceso de diseño porque no es una característica separada que se pueda agregar después. Al igual que con otras funciones de aplicación, las decisiones que se tomen durante las primeras fases de diseño y codificación determinarán en gran medida la escalabilidad de la aplicación.

La escalabilidad de una aplicación requiere una pertenencia equilibrada entre dos dominios distintos, software y hardware. Puede avanzar grandes pasos que aumenten la escalabilidad de un dominio sólo para sabotearlos cometiendo errores en el otro. Por ejemplo, la creación de un grupo de servidores Web con equilibrio de carga no beneficiará una aplicación Web que se ha diseñado para ejecutarse en un solo equipo. De igual modo, el diseño de una aplicación altamente escalable y su implementación en equipos conectados a una red con poco ancho de banda no controlará bien las cargas pesadas cuando se sature el tráfico en la red.

Puesto que la escalabilidad no es un problema de diseño de las aplicaciones independientes, aquí se tratan las aplicaciones distribuidas. Las aplicaciones distribuidas están también un paso más allá de las tradicionales aplicaciones de cliente-servidor. Las aplicaciones distribuidas son aplicaciones que están diseñadas como aplicaciones de n niveles. La arquitectura de estas aplicaciones distribuidas favorece el diseño de aplicaciones escalables compartiendo recursos, como bases de datos y componentes empresariales.

Escalar en vertical

El escalado en vertical es el término que más se utiliza para lograr escalabilidad utilizando software mejor, más rápido y más caro. El escalado incluye agregar más memoria, más procesadores o procesadores más rápidos o, simplemente, migrar la aplicación a un único equipo más potente. Normalmente, este método permite un aumento en la capacidad sin requerir cambios en el código fuente. Desde el punto de vista administrativo, las cosas permanecen igual puesto que sigue habiendo un único equipo que administrar.



Escalar en vertical

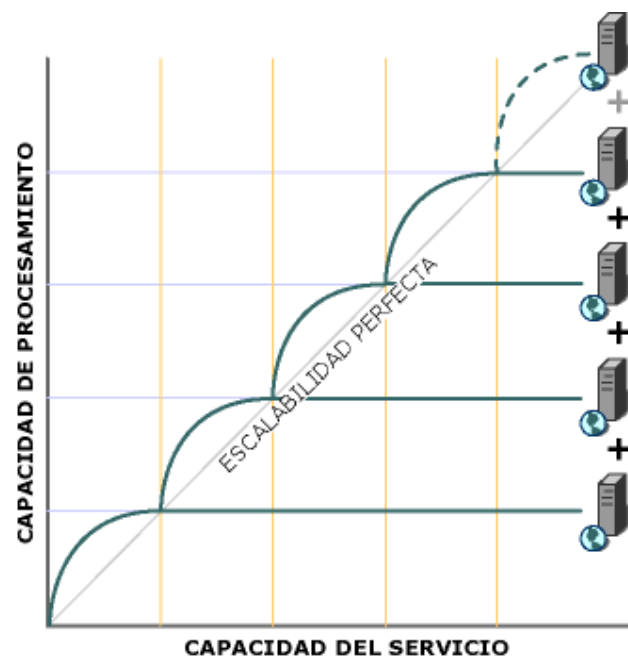
Actualizar un componente de hardware en un equipo sólo mueve el límite de capacidad de procesamiento de un lado a otro. Por ejemplo, una máquina que está al 100 % de uso de la

CPU podría mejorar su capacidad agregando otra CPU. Sin embargo, la limitación puede pasar de la CPU a la memoria del sistema. Agregar CPU no aporta rendimiento en un modelo lineal. En su lugar, el rendimiento va disminuyendo cada vez que se agrega un procesador adicional. Para equipos con configuraciones de varios procesadores simétricos (SMP), cada procesador adicional produce una sobrecarga del sistema. Por tanto, un equipo con cuatro procesadores no obtendrá una mejora del 400% en capacidad sobre una versión con un único procesador. Una vez que haya actualizado todos los componentes de hardware al máximo de su capacidad, llegará el momento en que alcance el límite real de la capacidad de procesamiento del equipo. Llegado ese punto, el siguiente paso es escalar en vertical para moverse a otro equipo.

Escalar en vertical conlleva también otros posibles problemas. El uso de un único equipo en el que basar una aplicación crea un único punto de error, lo que disminuye enormemente la tolerancia de errores del sistema. Si bien algunos métodos, como varias fuentes de alimentación, pueden implementar redundancia en un sistema de un único equipo, pueden resultar costosas.

Escalar en horizontal

Una alternativa a escalar en vertical es escalar en horizontal. Escalar en horizontal aprovecha el ahorro que supone utilizar el hardware de PC activo para distribuir la carga de procesamiento en más de un servidor. Aunque el escalado en horizontal se logra utilizando muchos equipos, la colección funciona esencialmente como un único equipo. Al dedicar varios equipos a una tarea común, mejora la tolerancia de errores de la aplicación. Por supuesto, desde el punto de vista del administrador, escalar en horizontal presenta un desafío mayor de administración debido al mayor número de equipos.



Escalar en horizontal

Los desarrolladores y administradores utilizan una gran variedad de técnicas de equilibrio de carga para escalar en horizontal con la plataforma Windows. El equilibrio de carga permite escalar un sitio en horizontal a través de un clúster de servidores, facilitando la adición de capacidad agregando más servidores duplicados. También proporciona redundancia, concediendo al sitio capacidad de recuperación de conmutación por error, de manera que



permanece disponible para los usuarios incluso si uno o más servidores fallan (o si es preciso retirarlos del servicio para realizar labores de mantenimiento). El escalado en horizontal proporciona un método de escalabilidad que no se ve mermado por limitaciones de hardware. Cada servidor adicional proporciona una mejora casi lineal de la escalabilidad.

La clave para escalar horizontalmente una aplicación con éxito es la transparencia de ubicación. Si alguna parte del código de la aplicación depende de saber qué servidor está ejecutando el código, no se ha logrado la transparencia de ubicación y será difícil el escalado en horizontal. Esta situación se denomina afinidad de ubicación. La afinidad de ubicación requiere cambios de código para escalar una aplicación en horizontal de un servidor a varios, lo que, en pocas ocasiones, constituye una opción económica. Si diseña la aplicación con transparencia de ubicación en mente, resulta más fácil escalarla en horizontal.

Un buen diseño es la base de una aplicación altamente escalable. En ningún otro momento de la vida de una aplicación puede tener mayor impacto una decisión sobre la escalabilidad de la aplicación que durante la fase de diseño.



Pirámide de escalabilidad

Como indica la pirámide de escalabilidad, hardware, software y ajuste rápidos son sólo una pequeña parte de la ecuación de escalabilidad. En la base de la pirámide está el diseño, que tiene la mayor influencia en la escalabilidad. Conforme se desplaza en la pirámide a través de factores cada vez menos importantes, la capacidad de causar impacto en la escalabilidad disminuye. Lo que muestra la pirámide es que el diseño inteligente puede agregar más escalabilidad a una aplicación que el hardware.

Cuando se diseña para ofrecer escalabilidad, el principal objetivo es garantizar una administración eficaz de los recursos. El diseño para la escalabilidad no está limitado a ningún nivel o componente concreto de una aplicación. Los arquitectos de aplicaciones deben considerar la escalabilidad en todos los niveles, desde la interfaz de usuario hasta el almacén de datos. Los cinco mandamientos del diseño para la escalabilidad que se indican a continuación pueden ser muy útiles al realizar selecciones de diseño.

1.4.- Introducción a la plataforma .NET

En realidad .NET es mucho más que eso ya que ofrece un entorno gestionado de ejecución de aplicaciones, nuevos lenguajes de programación y compiladores, y permite el desarrollo de todo tipo de funcionalidades: desde programas de consola o servicios Windows hasta aplicaciones para dispositivos móviles, pasando por desarrollos de escritorio o para Internet.

Son estos últimos de los que nos ocuparemos en este curso. Pero antes conviene conocer los fundamentos en los que se basa cualquier aplicación creada con .NET, incluyendo las que nos interesan.

El entorno de ejecución CLR

.NET ofrece un entorno de ejecución para sus aplicaciones conocido como Common Language Runtime o CLR. La CLR es la implementación de Microsoft de un estándar llamado Common Language Infrastructure o CLI. Éste fue creado y promovido por la propia Microsoft pero desde hace años es un estándar reconocido mundialmente por el ECMA.

El CLR/CLI esencialmente define un entorno de ejecución virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET. Este entorno virtual se ocupa de multitud de cosas importantes para una aplicación: desde la gestión de la memoria y la vida de los objetos hasta la seguridad y la gestión de subprocesos.

Todos estos servicios unidos a su independencia respecto a arquitecturas computacionales convierten la CLR en una herramienta extraordinariamente útil puesto que, en teoría, cualquier aplicación escrita para funcionar según la CLI puede ejecutarse en cualquier tipo de arquitectura de hardware. Por ejemplo Microsoft dispone de implementación de .NET para Windows de 32 bits, Windows de 64 bits e incluso para Windows Mobile, cuyo hardware no tiene nada que ver con la arquitectura de un ordenador común.

1.5.- El Lenguaje Intermedio y el CLS

Al contrario que otros entornos, la plataforma .NET no está atada a un determinado lenguaje de programación ni favorece a uno determinado frente a otros. En la actualidad existen implementaciones para varias decenas de lenguajes que permiten escribir aplicaciones para la plataforma .NET. Los más conocidos son Visual Basic .NET, C# o J#, pero existen implementaciones de todo tipo, incluso de ¡COBOL!.

Lo mejor de todo es que cualquier componente creado con uno de estos lenguajes puede ser utilizado de forma transparente desde cualquier otro lenguaje .NET. Además, como ya se ha comentado, es posible ejecutar el código .NET en diferentes plataformas y sistemas operativos.

¿Cómo se consigue esta potente capacidad?

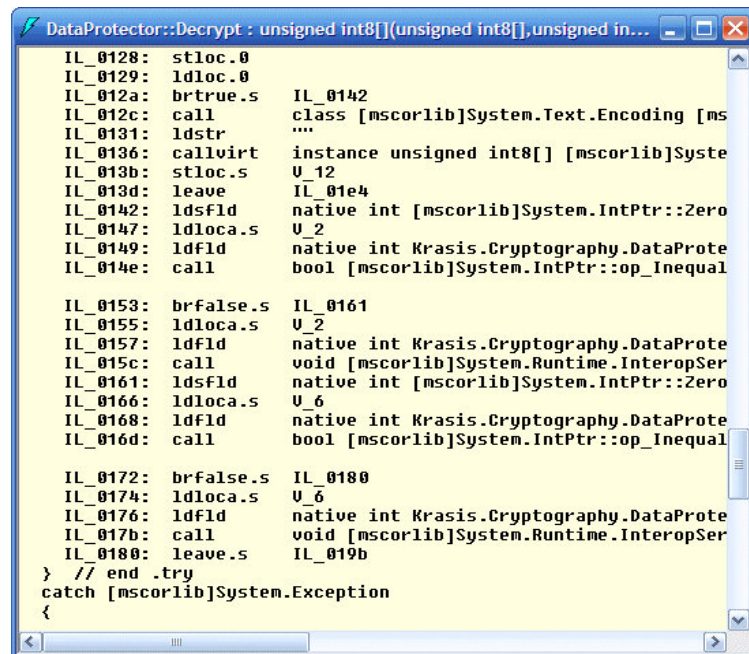
Dentro de la CLI, existe un lenguaje llamado IL (Intermediate Language o Lenguaje Intermedio) que está pensado de forma independiente al procesador en el que se vaya a ejecutar. Es algo parecido al código ensamblador pero de más alto nivel y creado para un hipotético procesador virtual que no está atado a una arquitectura determinada.

Cuando se compila una aplicación escrita en un lenguaje .NET cualquiera (da igual que sea VB, C# u otro de los soportados), el compilador lo que genera en realidad es un nuevo código escrito en este lenguaje intermedio. Así, todos los lenguajes .NET se usan como capa de más alto nivel para producir código IL.

Un elemento fundamental de la CLR es el compilador JIT (just-in-time). Su cometido es el de compilar bajo demanda y de manera transparente el código escrito en lenguaje intermedio a lenguaje nativo del procesador físico que va a ejecutar el código.

Al final, lo que se ejecuta es código nativo que ofrece un elevado rendimiento. Esto es cierto también para las aplicaciones Web escritas con ASP.NET y contrasta con las aplicaciones basadas en ASP clásico que eran interpretadas, no compiladas, y que jamás podrían llegar al nivel de desempeño que ofrece ASP.NET.

La siguiente figura muestra el aspecto que tiene el código intermedio de una aplicación sencilla y se puede obtener usando el desensamblador que viene con la plataforma .NET.



Código en lenguaje intermedio obtenido con ILDASM.exe

La especificación común de los lenguajes y el sistema de tipos comunes

Para conseguir la interoperabilidad entre lenguajes no sólo llega con el lenguaje intermedio, sino que es necesario disponer de unas "reglas del juego" que definan un conjunto de características que todos los lenguajes deben incorporar. A este conjunto regulador se le denomina Common Language Specification (CLS) o, en castellano, especificación común de los lenguajes.

Entre las cuestiones que regula la CLS se encuentran la nomenclatura, la forma de definir los miembros de los objetos, los metadatos de las aplicaciones, etc... Una de las partes más importantes de la CLS es la que se refiere a los tipos de datos.

Si alguna vez ha programado la API de Windows o ha tratado de llamar a una DLL escrita en C++ desde Visual Basic 6 habrá comprobado lo diferentes que son los tipos de datos de VB6 y de C++. Para evitar este tipo de problemas y poder gestionar de forma eficiente y segura el acceso a la memoria, la CLS define un conjunto de tipos de datos comunes (Common Type System o CTS) que indica qué tipos de datos se pueden manejar, cómo se declaran y se utilizan éstos y de qué manera se deben gestionar durante la ejecución.

Si nuestras bibliotecas de código utilizan en sus interfaces hacia el exterior datos definidos dentro de la CTS no existirán problemas a la hora de utilizarlos desde cualquier otro código escrito en la plataforma .NET.

Cada lenguaje .NET utiliza una sintaxis diferente para cada tipo de datos. Así, por ejemplo, el tipo común correspondiente a un número entero de 32 bits (System.Int32) se denomina Integer en Visual Basic .NET, pero se llama int en C#. En ambos casos representan el mismo tipo de datos que es lo que cuenta.

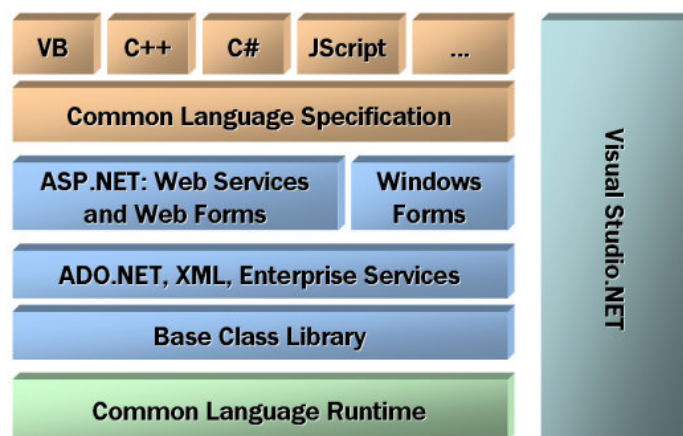
1.6.- La biblioteca de clases de .NET

Todo lo que se ha estado comentando hasta ahora en el curso constituye la base de la plataforma .NET. Si bien es muy interesante y fundamental, por sí mismo no nos serviría de mucho para crear programas si debiésemos crear toda la funcionalidad desde cero.

Obviamente esto no es así, y la plataforma .NET nos ofrece infinidad de funcionalidades "de fábrica" que se utilizan como punto de partida para crear las aplicaciones. Existen funcionalidades básicas (por ejemplo todo lo relacionado con la E/S de datos o la seguridad) y funcionalidades avanzadas en las que se fundamentan categorías enteras de aplicaciones (acceso a datos, creación de aplicaciones Web...).

Toda esta funcionalidad está implementada en forma de bibliotecas de funciones que físicamente se encuentran en diversas DLL (bibliotecas de enlazado dinámico). A su conjunto se le denomina Base Classes Library (Biblioteca de clases base o BCL) y forman parte integral de la plataforma .NET, es decir, no se trata de añadidos que se deban obtener o adquirir aparte.

La siguiente figura ilustra la arquitectura conceptual de la plataforma .NET. En ella se pueden observar los elementos que se han mencionado en apartados anteriores (lenguajes, CLR, CLS...) y en qué lugar de se ubican las bibliotecas de clases base:



Distintos elementos de la plataforma .NET y cómo se relacionan entre sí.

Resulta muy útil para comprender lo explicado hasta ahora. No se preocupe si hay elementos que no conoce, más adelante los estudiaremos todos.

Todo lo que se encuentra en la BCL forma parte de la plataforma .NET. De hecho existe tal cantidad de funcionalidad integrada dentro de estas bibliotecas (hay decenas de miles de clases) que el mayor esfuerzo que todo programador que se inicia en .NET debe hacer es el aprendizaje de las más importantes. De todos modos Visual Studio ofrece mucha ayuda contextual (documentación, Intellisense...) y una vez que se aprenden los rudimentos resulta fácil ir avanzando en el conocimiento de la BCL a medida que lo vamos necesitando.

Los espacios de nombres

Dada la gran cantidad de clases que existen debe existir algún modo de organizarlas de un modo coherente. Además hay que tener en cuenta que podemos adquirir más funcionalidades (que se traducen en clases) a otros fabricantes, por no mencionar que crearemos continuamente nuevas clases propias.

Para solucionar este problema existen en todos los lenguajes .NET los espacios de nombres o namespaces.

Un espacio de nombres no es más que un identificador que permite organizar de modo estanco las clases que estén contenidas en él así como otros espacios de nombres.

Así, por ejemplo, todo lo que tiene que ver con el manejo de estructuras de datos XML en la plataforma .NET se encuentra bajo el espacio de nombres System.Xml. La funcionalidad fundamental para crear aplicaciones Web está en el espacio de nombres System.Web. Éste a



su vez contiene otros espacios de nombres más especializados como System.Web.Caching para la persistencia temporal de datos, System.Web.UI.WebControls, que contiene toda la funcionalidad de controles Web para interfaz de usuario, etc...

1.7.- Acceso a datos con ADO.NET

El acceso a fuentes de datos es algo indispensable en cualquier lenguaje o plataforma de desarrollo. La parte de la BCL que se especializa en el acceso a datos se denomina de forma genérica como ADO.NET.

Si usted ha programado con Visual Basic 6.0 o con ASP, ha empleado en su código con total seguridad la interfaz de acceso a datos conocida como ADO (ActiveX Data Objects), puede que combinado con ODBC (Open Database Connectivity).

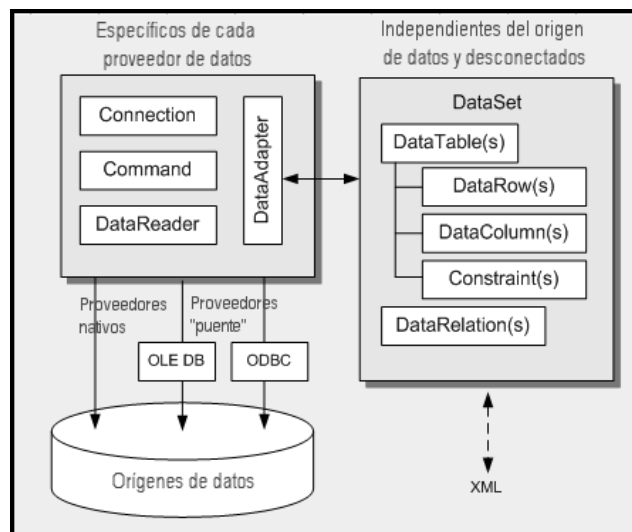
ADO.NET ofrece una funcionalidad completamente nueva, que tiene poco que ver con lo existente hasta la fecha en el mercado. Sin embargo, con el ánimo de retirar barreras a su aprendizaje, Microsoft denominó a su nuevo modelo de acceso a datos con un nombre similar y algunas de sus clases recuerdan a objetos de propósito análogo en el vetusto ADO.

ADO.NET es un modelo de acceso mucho más orientado al trabajo desconectado de las fuentes de datos de lo que nunca fue ADO. Si bien este último ofrecía la posibilidad de desconectar los Recordsets y ofrecía una forma de serialización de estos a través de las diferentes capas de una aplicación, el mecanismo no es ni de lejos tan potente como el que nos ofrece ADO.NET.

El objeto más importante a la hora de trabajar con el nuevo modelo de acceso a datos es el DataSet. Sin exagerar demasiado podríamos calificarlo casi como un motor de datos relacionales en memoria. Aunque hay quien lo asimila a los clásicos Recordsets su funcionalidad va mucho más allá como se verá en el correspondiente módulo.

Arquitectura de ADO.NET

El concepto más importante que hay que tener claro sobre ADO.NET es su modo de funcionar, que se revela claramente al analizar su arquitectura:



Arquitectura de ADO.NET

Existen dos capas fundamentales dentro de su arquitectura: la **capa conectada** y la **desconectada**.

Capa conectada

La primera de ellas contiene objetos especializados en la conexión con los orígenes de datos. Así, la clase genérica **Connection** se utiliza para establecer conexiones a los orígenes de datos. La clase **Command** se encarga de enviar comandos de toda índole al origen de datos. Por fin la clase **DataReader** está especializada en leer los resultados de los comandos mientras se permanece conectado al origen de datos.

La clase **DataAdapter** hace uso de las tres anteriores para actuar de puente entre la capa conectada y la desconectada.

Estas clases son abstractas, es decir, no tienen una implementación real de la que se pueda hacer uso directamente. Es en este punto en donde entran en juego los **proveedores de datos**. Cada origen de datos tiene un modo especial de comunicarse con los programas que los utilizan, además de otras particularidades que se deben contemplar. Un proveedor de datos de ADO.NET es una implementación concreta de las clases conectadas abstractas que hemos visto, que hereda de éstas y que tiene en cuenta ya todas las particularidades del origen de datos en cuestión.

Así, por ejemplo, las clases específicas para acceder a SQL Server se llaman **SqlConnection**, **SqlCommand**, **SqlDataReader** y **SqlDataAdapter** y se encuentran bajo el espacio de nombres **System.Data.SqlClient**. Es decir, al contrario que en ADO clásico no hay una única clase **Connection** o **Command** que se use en cada caso, si no que existen clases especializadas para conectarse y recuperar información de cada tipo de origen de datos.

Existen **proveedores nativos**, que son los que se comunican directamente con el origen de datos (por ejemplo el de SQL Server o el de Oracle), y **proveedores "puente"**, que se utilizan para acceder a través de ODBC u OLEDB cuando no existe un proveedor nativo para un determinado origen de datos.

Capa desconectada

Una vez que ya se han recuperado los datos desde cualquier origen de datos que requiera una conexión ésta ya no es necesaria. Sin embargo sigue siendo necesario trabajar con los datos obtenidos de una manera flexible. Es aquí cuando la capa de datos desconectada entra en juego. Además, en muchas ocasiones es necesario tratar con datos que no han sido obtenidos desde un origen de datos relacional con el que se requiera una conexión. A veces únicamente necesitamos un almacén de datos temporal pero que ofrezca características avanzadas de gestión y acceso a la información.

Por otra parte las conexiones con las bases de datos son uno de los recursos más escasos con los que contamos al desarrollar. Su mala utilización es la causa más frecuente de cuellos de botella en las aplicaciones y de que éstas no escalen como es debido. Esta afirmación es especialmente importante en las aplicaciones Web en las que se pueden recibir muchas solicitudes simultáneas de cualquier parte del mundo.

Finalmente otro motivo por el que es importante el uso de los datos desconectado de su origen es la transferencia de información entre capas de una aplicación. Éstas pueden encontrarse distribuidas por diferentes equipos, e incluso en diferentes lugares del mundo gracias a Internet. Por ello es necesario disponer de algún modo genérico y eficiente de poder transportar los datos entre diferentes lugares, utilizarlos en cualquiera de ellos y posteriormente tener la capacidad de conciliar los cambios realizados sobre ellos con el origen de datos del que proceden.

Todo esto y mucho más es lo que nos otorga el uso de los objetos **DataSet**. Es obvio que no se trata de tareas triviales, pero los objetos **DataSet** están pensados y diseñados con estos objetivos en mente. Como podremos comprobar más adelante en este curso es bastante sencillo conseguir estas funcionalidades tan avanzadas y algunas otras simplemente usando de manera adecuada este tipo de objetos.

Los *DataSet*, como cualquier otra clase no sellada de .NET, se pueden extender mediante herencia. Ello facilita una técnica avanzada que consiste en crear tipos nuevos de *DataSet* especializados en la gestión de una información concreta (por ejemplo un conjunto de tablas relacionadas). Estas nuevas tipos clases se denominan genéricamente ***DataSet Tipados***, y permiten el acceso mucho más cómodo a los datos que representan, verificando reglas de negocio, y validaciones de tipos de datos más estrictas.

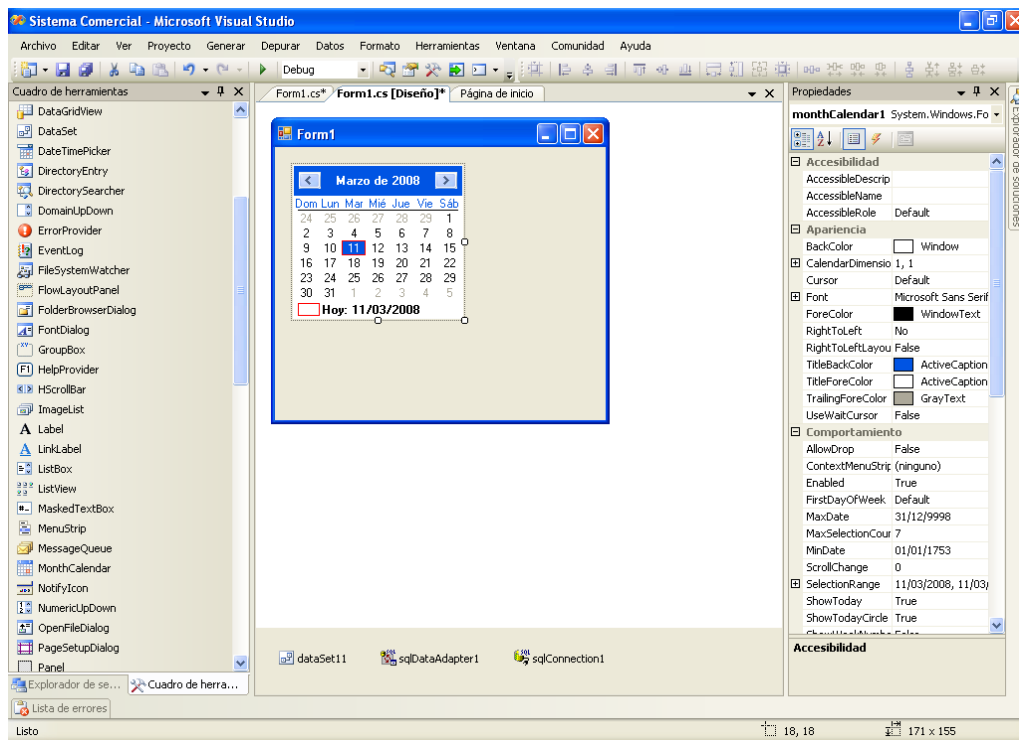
1.8.- Aplicaciones Windows Forms

Las aplicaciones de escritorio son aquellas basadas en ventanas y controles comunes de Windows que se ejecutan en local. Son el mismo tipo de aplicaciones que antes construiríamos con Visual Basic 6 u otros entornos similares.

En la plataforma .NET el espacio de nombres que ofrece las clases necesarias para construir aplicaciones de escritorio bajo Windows se denomina ***Windows Forms***. Este es también el nombre genérico que se le otorga ahora a este tipo de programas basados en ventanas.

Windows Forms está constituido por multitud de clases especializadas que ofrecen funcionalidades para el trabajo con ventanas, botones, rejillas, campos de texto y todo este tipo de controles habituales en las aplicaciones de escritorio.

Visual Studio ofrece todo lo necesario para crear visualmente este tipo de programas, de un modo similar aunque más rico al que ofrecía el entorno de desarrollo integrado de Visual Basic.

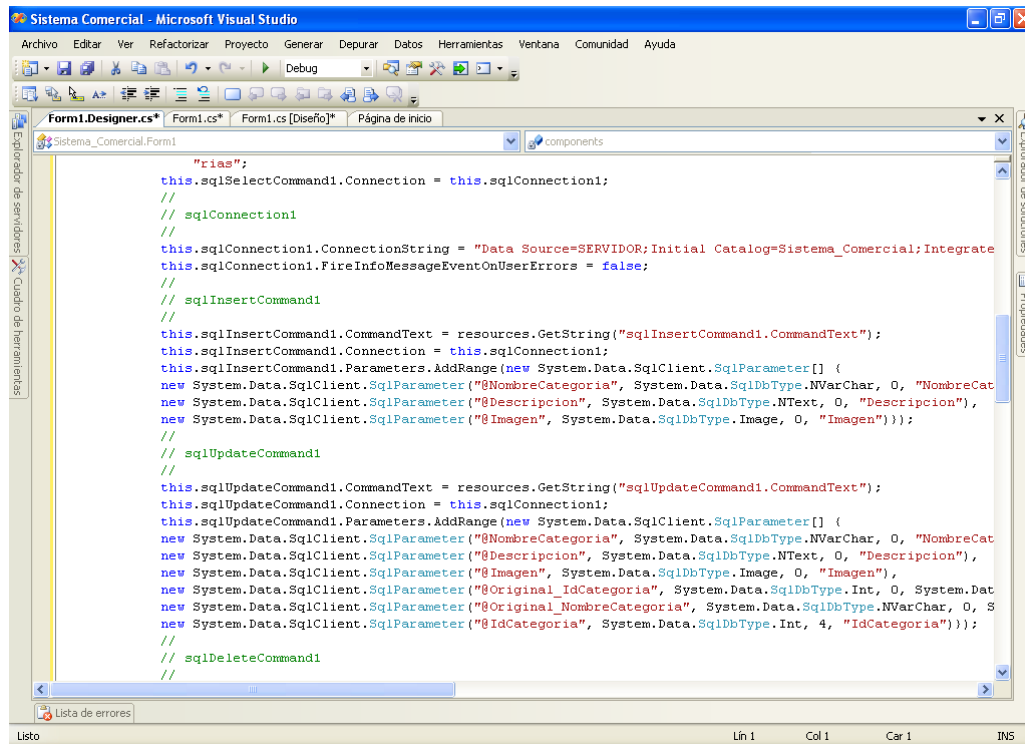


Diseñador de interfaces de aplicaciones de escritorio con Windows Forms en Visual Studio 2005.

Al contrario que en VB6, .NET proporciona control sobre todos los aspectos de las ventanas y controles, no dejando nada fuera del alcance del programador y otorgando por lo tanto la máxima flexibilidad. Los formularios (ventanas) son clases que heredan de la clase base ***Form***, y cuyos controles son miembros de ésta. De hecho se trata únicamente de código y no

es necesario (aunque sí muy recomendable) emplear el diseñador gráfico de Visual Studio para crearlas.

Este es el aspecto que presenta parte del código que genera la interfaz mostrada en la anterior figura:



Código autogenerated por Visual Studio para crear la interfaz de la figura anterior.

Al contrario que en Visual Basic tradicional, en donde siempre existían instancias por defecto de los formularios que podíamos usar directamente, en .NET es necesario crear un objeto antes de poder hacer uso de los formularios:

```
Dim frm As New MiFormulario
frm.Show()
```

Todos los controles heredan de una clase **Control** por lo que conservan una serie de funcionalidades comunes muy interesantes, como la capacidad de gestionarlos en el diseñador (moviéndolos, alineándolos...), de definir márgenes entre ellos o hacer que se adapten al tamaño de su contenedor.

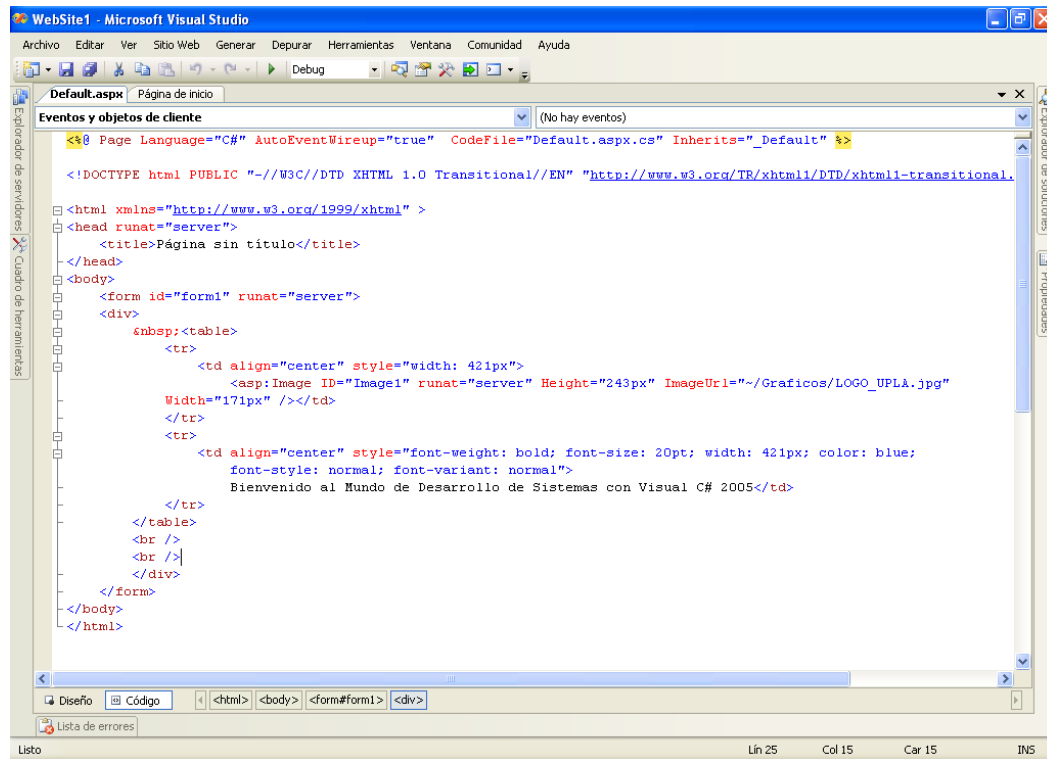
1.9.- Aplicaciones Web Forms

Tradicionalmente las aplicaciones Web se han desarrollado siguiendo un modelo mixto que intercalaba código HTML y JavaScript propio de páginas Web (parte cliente), junto con código que se ejecutaría en el servidor (parte servidora). Este modelo contrastaba por completo con el modelo orientado a eventos seguido por las principales herramientas de desarrollo de aplicaciones de escritorio.

En el modelo orientado a eventos se define la interfaz de usuario colocando controles en un contenedor y se escribe el código que actuará como respuesta a las interacciones de los usuarios sobre estos controles. Si conoce el diseñador de VB6 o de Windows Forms mencionado en el apartado anterior sabe exactamente a qué nos referimos.

Hacer esto en una aplicación de escritorio no tiene mayor dificultad ya que todo el código se ejecuta en el mismo lugar. La principal característica de las aplicaciones Web sin embargo es que se la interfaz de usuario (lo que los usuarios de la aplicación ven) se ejecuta en un lugar diferente al código de la aplicación que reside en un servidor. Para mayor desgracia estas aplicaciones se basan en el uso del protocolo HTTP que es un protocolo sin estado y que no conserva la conexión entre dos llamadas consecutivas.

Por ejemplo, el siguiente código ilustra el código que es necesario escribir en ASP para disponer de una página que rellena una lista de selección con unos cuantos nombres (podrían salir de una base de datos y aún sería más complicado), y que dispone de un botón que escribe un saludo para el nombre que se haya elegido de la lista.



Código ASP sencillo que genera una lista de selección y saluda al presionar un botón.

Obviamente se podría haber simplificado sin enviar el formulario al servidor usando JavaScript en el cliente para mostrar el saludo, pero la intención es ilustrar la mezcla de código de cliente y de servidor que existe en este tipo de aplicaciones.

Las principales desventajas de este tipo de codificación son las siguientes:

1. **No existe separación entre el diseño y la lógica de las aplicaciones.** Si queremos cambiar sustancialmente la apariencia de la aplicación Web lo tendremos bastante complicado puesto que el código del servidor está mezclado entre el HTML.
2. **En ASP clásico no existe el concepto de control** para la interfaz de usuario. Lo único que hay es HTML y JavaScript que se deben generar desde el servidor. En el ejemplo de la figura para generar un control de lista con unos elementos no podemos asignar una propiedad de la lista (porque no existe tal lista), sino que tenemos que crear un bucle que genere los elementos HTML necesarios para generarla. Tampoco disponemos de un diseñador visual que nos permita gestionar los controles y elementos

HTML existentes, y menos cuando éstos se encuentran mezclados con el código del servidor.

3. **No disponemos de forma de detectar en el servidor que se ha realizado algo en el cliente.** El cliente se encuentra desconectado desde el momento en que se termina de devolver la página. Sólo se recibe información en el servidor cuando se solicita una nueva página o cuando se envía un formulario tal y como se hace en el ejemplo, debiéndonos encargar nosotros de averiguar si la petición es la primera vez que se hace o no, y de dar la respuesta adecuada. En cualquier caso es mucho menos intuitivo que el modelo de respuesta a eventos de una aplicación de escritorio.
4. **No existe constancia del estado de los controles de cada página entre las llamadas.** En cada ejecución de la página tendremos que recrear completamente la salida. Por ejemplo si presionamos el botón “Di Hola” tenemos que escribir además de la etiqueta “Hola, nombre” el resto de la pantalla, incluyendo la lista con todos los nombres dejando seleccionado el mismo que hubiese antes. Si estos nombres viniesen de una base de datos esto puede ser todavía más ineficiente y tendremos que buscar métodos alternativos para generarlos ya que en ASP tampoco se deben almacenar en los objetos de sesión y/o aplicación Recordsets resultado de consultas.
5. **No existe el concepto de Propiedad de los controles.** En una aplicación Windows asignamos el texto de un campo usando una propiedad (por ejemplo Text1.Text = “Hola”) y ésta se asigna y permanece en la interfaz sin que tengamos que hacer nada. En una aplicación Web clásica tenemos que almacenarlas en algún sitio (una variable de sesión o un campo oculto) para conservarlas entre diferentes peticiones de una misma página.
6. **Los controles complejos no tienen forma de enviar sus valores al servidor.** Si intentamos crear una interfaz avanzada que utilice tablas y otros elementos que no son controles de entrada de datos de formularios de HTML tendremos que inventarnos mecanismos propios para recoger esos datos y enviarlos al servidor.

La principal aportación de ASP.NET al mundo de la programación es que ha llevado a la Web el paradigma de la programación orientada a eventos propia de aplicaciones de escritorio, ofreciendo:

- Separación entre diseño y lógica.
- Componentes de interfaz de usuario, tanto estándar como de terceras empresas o propios.
- Diseñadores gráficos.
- Eventos.
- Estado.
- Enlazado a datos desde la interfaz.



RESUMEN

Desarrollar sistemas de información para organizaciones utilizando herramientas Microsoft propone a los programadores participar e integrarse a las nuevas formas eficientes de desarrollo de sistemas. Visual Studio 2005 y SQL Server 2005 con todo el conjunto de aplicaciones integradas reduce la complejidad de creación de soluciones orientadas a servicios, facilita la comunicación entre miembros de un equipo de desarrollo y contribuye a crear sistemas integrados.

**BIBLIOGRAFÍA RECOMENDADA**

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.

**NEXO**

La unidad TEMÁTICA desarrollada brinda un panorama general de los criterios fundamentales a tomar en cuenta en el desarrollo de un sistema de información. Como también de las herramientas de actualidad que permiten llevar a cabo esta implementación.

**ACTIVIDAD**

El presente manual incluye dos DVDs que contienen los programas de Microsoft Visual Studio y Microsoft SQL Server en sus versiones más recientes con licencia de uso por 90 días. Usted deberá cambiar el nombre de su computadora a **SERVIDOR** e instalar los programas en el siguiente orden: Microsoft SQL Server - Microsoft Visual Studio, el mismo que le permitirá poner en práctica todo el conjunto de programas y aplicaciones incluidas en el curso.

También se le proporciona una copia de seguridad de la base de datos utilizado en los diferentes programas del presente curso, Usted deberá restaurarla en el programa Microsoft SQL Server de su computador.

**AUTOEVALUACIÓN FORMATIVA**

Dada la naturaleza de la asignatura usted deberá desarrollar un sistema de información para una organización. Elija una de acuerdo a su preferencia y responda a las siguientes preguntas:

- 1.- ¿Qué porcentaje de disponibilidad será el adecuado para una aplicación que se implementará en la organización? Establezca en forma aleatoria un número determinado de errores.
- 2.- Determine la confiabilidad que debería tener el sistema con los mismos datos del ejercicio anterior.

Unidad Temática II

CARACTERÍSTICAS DEL LENGUAJE C#

2.1.- El sistema de Tipos

En esta lección veremos los tipos de datos que .NET Framework pone a nuestra disposición y cómo tendremos que usarlos desde Visual C# 2005.

A continuación daremos un repaso a conceptos básicos o elementales sobre los tipos de datos, que aunque puede que nos sean familiares, es importante que lo veamos para poder comprender mejor cómo están definidos y organizados los tipos de datos en .NET Framework.

Tipos de datos de .NET

Visual C# 2005 está totalmente integrado con .NET Framework, por tanto los tipos de datos que podremos usar con este lenguaje serán los definidos en este "marco de trabajo", por este motivo vamos a empezar usando algunas de las definiciones que nos encontraremos al recorrer la documentación que acompaña a este lenguaje de programación.

Los tipos de datos que podemos usar en Visual C# 2005 son los mismos tipos de datos definidos en .NET Framework y por tanto están soportados por todos los lenguajes que usan esta tecnología. Estos tipos comunes se conocen como el Common Type System, (CTS), que traducido viene a significar el sistema de tipos comunes de .NET. El hecho de que los tipos de datos usados en todos los lenguajes .NET estén definidos por el propio Framework nos asegura que independientemente del lenguaje que estemos usando, siempre utilizaremos el mismo tipo interno de .NET, si bien cada lenguaje puede usar un nombre (o alias) para referirse a ellos, aunque lo importante es que siempre serán los mismos datos, independientemente de cómo se llame en cada lenguaje. Esto es una gran ventaja, ya que nos permite usarlos sin ningún tipo de problemas para acceder a ensamblados creados con otros lenguajes, siempre que esos lenguajes sean compatibles con los tipos de datos de .NET.

Tipos primitivos

Veamos en la siguiente tabla los tipos de datos definidos en .NET Framework y los alias utilizados en Visual C# 2005.

.NET Framework	C#
System.Boolean	bool
System.Byte	byte
System.Int16	short
System.Int32	int
System.Int64	long
System.Single	float
System.Double	double
System.Decimal	decimal
System.Char	char
System.String	string
System.Object	object
System.DateTime	N.A. (DateTime)
System.SByte	sbyte
System.UInt16	ushort
System.UInt32	uint

System.UInt64	ulong
---------------	-------

Tipos de datos de .NET y su equivalencia en C#

El único tipo de datos que no tiene un "alias" en C# es tipo DateTime, por tanto siempre usaremos el definido en el propio .NET. Debemos tener en cuenta, al menos si el rendimiento es una de nuestras prioridades, que las cadenas en .NET son inmutables, es decir, una vez que se han creado no se pueden modificar y en caso de que queramos cambiar el contenido, .NET se encarga de desechar la anterior y crear una nueva cadena, por tanto si usamos las cadenas para realizar concatenaciones (unión de cadenas para crear una nueva), el rendimiento será muy bajo, si bien existe una clase en .NET que es ideal para estos casos y cuyo rendimiento es superior al tipo string: la clase StringBuilder.

Las últimas filas mostradas en la tabla son tipos especiales que si bien son parte del sistema de tipos comunes (CTS) no forman parte de la Common Language Specification (CLS), es decir la especificación común para los lenguajes "compatibles" con .NET, por tanto, si queremos crear aplicaciones que puedan interoperar con todos los lenguajes de .NET, esos tipos no debemos usarlos como valores de devolución de funciones ni como tipo de datos usado en los parámetros de las funciones.

Los tipos mostrados en la tabla son los tipos primitivos de .NET y por extensión de Visual C# 2005, es decir son tipos "elementales" para los cuales cada lenguaje define su propia palabra clave equivalente con el tipo definido en el CTS de .NET Framework. Todos estos tipos primitivos podemos usarlos tanto por medio de los tipos propios de Visual C#, los tipos definidos en .NET o bien como literales. Por ejemplo, podemos definir un número entero largo (long) literal indicándolo con el sufijo L: 12345L o bien asignándolo a un valor de tipo long o a un tipo Sytem.Int64 de .NET. La única excepción de los tipos mostrados en la tabla es el tipo de datos object, este es un caso especial del que nos ocuparemos en la próxima lección.

Sufijos o caracteres y símbolos identificadores para los tipos

Cuando usamos valores literales numéricos en Visual C# 2005, el tipo de datos que le asigna el compilador es el que más se acerque al tipo en el que lo estamos asignando, es decir, el propio compilador de C# hace lo que se conoce como "promoción de tipos", esto significa que si estamos asignando un valor a una variable de tipo entero, el compilador comprobará que tipo es el más adecuado, usando este orden: int, uint, long, ulong. Si el tipo al que queremos asignar el valor es un tipo diferente a los cuatro que hemos mostrado, el compilador nos avisará que el tipo de datos que estamos usando no es el correcto.

Por otro lado, si la constante contiene el separador de decimales, el compilador de C# asumirá que estamos trabajando con valores de coma flotante, pero siempre asumirá que el valor es de tipo double. Por tanto si el tipo que hay a la izquierda de esa constante no es double, el compilador mostrará un error.

Estos errores de compilación los podemos evitar de dos formas:

La primera es usando un sufijo en la constante, con ese sufijo le indicamos que el tipo de datos que queremos que tenga esa constante. En la siguiente tabla tenemos los sufijos que podemos usar en C#.

La segunda forma de evitar estos errores de compilación, es haciendo una conversión explícita de tipos. En la sección siguiente veremos algunos ejemplos de conversiones de tipos.

Tipo de datos	Sufijo de C#
short	N.A.
int	N.A.
long	L
float	F
double	D
decimal	M
ushort	N.A.
uint	UI

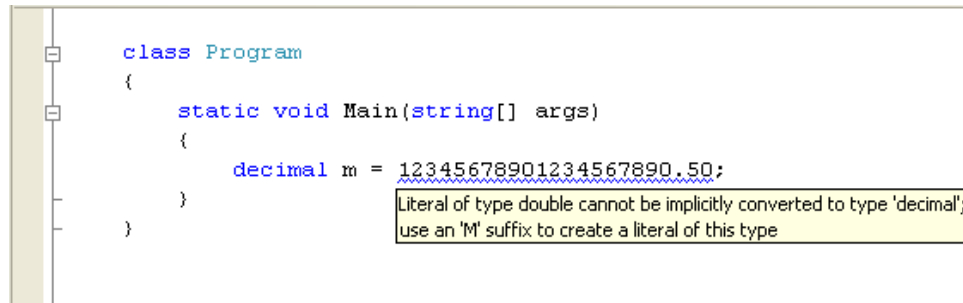
ulong	UL
string	N.A.

Sufijos para identificar los tipos de datos

Los indicados con N.A. es que no existe un sufijo para ese tipo.

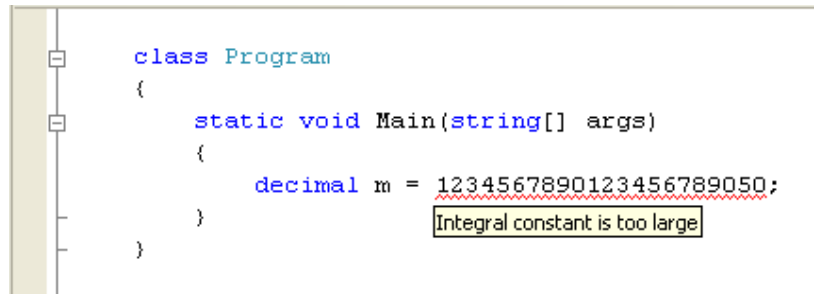
El uso de estos caracteres nos puede resultar de utilidad particularmente para los tipos de datos que no se pueden convertir de forma automática.

Por ejemplo, si queremos asignar este valor literal **12345678901234567890.50** a un tipo *decimal*, el compilador de Visual C# nos avisará de que no se puede asignar un valor doble a uno decimal, y nos recomienda que usemos el sufijo **M** en el literal.



Error de compilación al asignar un valor double a una variable decimal

Tal como indicamos en la nota, el entorno de desarrollo integrado (IDE) de Visual C# 2005 nos avisará de los errores de compilación una vez que hayamos compilado el código, aunque los errores sintácticos los mostrará justo al producirse. Estos errores sintácticos los mostrará con los dientes de sierra de color rojo, tal como podemos comprobar en la siguiente figura al intentar asignar un valor entero que es demasiado grande.



Error sintáctico y resalte en el IDE de Visual C#

Promociones numéricas implícitas (automáticas)

Como hemos comentado anteriormente, el compilador de C# siempre intentará usar el valor más adecuado cuando estamos utilizando distintos tipos de datos.

Esas promociones, (convertir un valor de un tipo pequeño en otro de mayor capacidad), las realizará solamente en los tipos de datos que no produzcan pérdida de información o que están contempladas en el compilador. Por ejemplo, podemos asignar un valor de tipo *short* a un tipo *long*, ya que no se producirá ningún tipo de pérdida de información, porque el valor de un *short* es menor que el de un *long*.

En la siguiente tabla tenemos las conversiones implícitas (o automáticas) que puede hacer el compilador de Visual C#.

Del tipo	Al tipo (o tipos)
sbyte	short, int, long, float, double, o decimal
byte	short, ushort, int, uint, long, ulong, float, double, o decimal
short	int, long, float, double, o decimal
ushort	int, uint, long, ulong, float, double, o decimal
int	long, float, double, o decimal
uint	long, ulong, float, double, o decimal
long	float, double, o decimal
char	ushort, int, uint, long, ulong, float, double, o decimal
float	double
ulong	float, double, o decimal

Conversiones implícitas

En esta tabla se incluye el tipo *char*, ya que aunque un *char* representa un carácter, realmente es un tipo entero. A pesar de que podamos convertir un *char* en un tipo numérico, no podemos hacer lo contrario, al menos de forma implícita.

Conversiones numéricas explícitas

Hay ocasiones en las que el compilador considera que no se pueden hacer ciertas conversiones o asignaciones entre variables o valores literales de distintos tipos de datos. Pero si nosotros sabemos que es posible hacerlas, podemos obligar al compilador a que las acepte. De esa forma no recibiremos ningún error y la aplicación se creará correctamente. Otra cosa es que cuando el programa esté en ejecución esa conversión que hemos hecho de forma explícita, es decir, obligando al compilador a hacerla, (ahora veremos cómo podemos obligar al compilador a que cumpla nuestras órdenes), puede fallar, por la sencilla razón de que no se pueda hacer.

Para hacer una conversión explícita, lo haremos de esta forma:

```
int entero = (int)valor_long;
```

Es decir, incluimos entre paréntesis el tipo de datos al que queremos convertir y a continuación indicamos el valor a convertir.

¿Cuándo debemos usar las conversiones explícitas?

Las conversiones explícitas las tenemos que usar en todos los casos que no se adecuen a los casos mostrados en la tabla de conversiones implícitas.

¿Siempre funcionan las conversiones explícitas?

Habrán situaciones en que una conversión explícita produzca un error de desbordamiento o bien se asigne el valor máximo permitido por el tipo de destino desestimando el resto del valor asignado. Esto dependerá de ciertas circunstancias, dependiendo del tipo de datos que intervengan en la conversión y de que usemos ciertas instrucciones de C# o de la utilización de opciones del compilador.

Cuando convertimos un literal de tipo entero de mayor capacidad en uno que tiene menos, el compilador producirá un error, porque "sabe" que el valor grande no "cabe" en el pequeño.



Este comportamiento lo podemos cambiar de forma que si hacemos ese tipo de conversión, sólo se almacene en el tipo de menor capacidad el valor que pueda soportar, desechando el resto, pero debido a que esto puede producir pérdida de información, hay que hacerlo de forma explícita, indicándolo con la instrucción `unchecked`.

En el siguiente ejemplo, se producirá un error de compilación indicándonos que el valor que queremos almacenar es muy grande para el tipo de destino:

```
short sh = (short)1234567890;
```

Para solucionarlo debemos usar la instrucción `unchecked`:

```
short sh = unchecked((short)1234567890);
```

Si en lugar de trabajar con un literal lo hacemos con una variable, el compilador no sabe que valor contiene la variable, por tanto compilará sin problemas, aunque ese error de desbordamiento se puede producir en tiempo de ejecución.

En este caso concreto o en otros en los que en la conversión intervengan variables de tipos enteros (`int`, `short`, `byte`, `long`, etc.), no se producirá ningún error de compilación ni en tiempo de ejecución, el valor que se almacenará en el entero de menor capacidad será el valor máximo que dicho tipo pueda contener, todo esto sin necesidad de usar la instrucción `unchecked`, ya que ese es el comportamiento predeterminado del compilador, para cambiarlo, debemos usar la instrucción `checked` o bien usar la opción `/checked+` del compilador.

De forma predeterminada, (usando `unchecked`), si en esas conversiones explícitas intervienen tipos decimales o de punto flotante el comportamiento dependerá del tipo de datos que reciba el valor. Asignar un valor de tipo flotante a uno entero no producirá nunca un error de desbordamiento (`overflow`), pero si el valor a convertir es un tipo decimal, se producirá un error si el contenido de ese valor decimal es mayor que el soportado por el tipo entero al que queremos asignarlo.

Por ejemplo, si utilizamos este código, todo funcionará bien, porque el valor de `m` es adecuado para un valor `int`:

```
decimal m = 123456.55M;  
int i = (int)m;
```

Pero si tenemos este otro código, se producirá un error en tiempo de ejecución, ya que el valor de `m` es mayor que el soportado por un valor de tipo `int`:

```
decimal m = 12345678901234567890.55M;  
int i = (int)m;
```

Funciones de conversión

En C# no existen funciones de conversión como instrucciones independientes, siempre se utiliza el formato mostrado; en el argot se utiliza la expresión inglesa utilizada para indicar que hacemos una conversión: `cast`.

Aunque C# no dispone de funciones de conversión, .NET Framework si que las tiene, además de que los propios tipos de datos soportan algunas, por ejemplo, para convertir un valor numérico en uno de tipo cadena, podemos usar el método `ToString()` que todos los tipos de .NET tienen. Pero si queremos realizar otras conversiones entre distintos tipos numéricos, usaremos la clase `Convert`, la cual tiene funciones para realizar conversiones entre los tipos numéricos soportados por .NET:

```
i = Convert.ToInt32(m);
```

Tipos por valor y tipos por referencia

Los tipos de datos de .NET los podemos definir en dos grupos:

- Tipos por valor
- Tipos por referencia

Los tipos por valor son tipos de datos cuyo valor se almacena en la pila o en la memoria "cercana", como los numéricos que hemos visto. Podemos decir que el acceso al valor contenido en uno de estos tipos es directo, es decir se almacena directamente en la memoria reservada para ese tipo y cualquier cambio que hagamos lo haremos directamente sobre dicho valor, de igual forma cuando copiamos valores de un tipo por valor a otro, estaremos haciendo copias independientes.

Por otro lado, los tipos por referencia se almacenan en el "monto" (heap) o memoria "lejana", a diferencia de los tipos por valor, los tipos por referencia lo único que almacenan es una referencia (o puntero) al valor asignado. Si hacemos copias de tipos por referencia, realmente lo que copiamos es la referencia propiamente dicha, pero no el contenido.

2.2.- Variables y constantes

Disponer de todos estos tipos de datos no tendría ningún sentido si no los pudiéramos usar de alguna otra forma que de forma literal. Y aquí es donde entran en juego las variables y constantes.

Definamos brevemente estos conceptos.

Las constantes son valores que nunca cambian y pueden ser de dos tipos:

Constantes literales, por ejemplo, cuando usamos 12345, estamos usando un valor constante, ya que ese número siempre tendrá el mismo valor.

Constantes con nombre, son constantes a las que les damos un nombre y tampoco pueden cambiar.

Por otro lado, las variables son como las constantes con nombres, pero su valor es variable, por tanto, puede que no siempre tengan el mismo valor, de ahí que se llamen *variables*.

Declarar constantes

Las constantes literales las usamos directamente, tal como hemos visto anteriormente, pero para usar las constantes con nombre debemos declararlas previamente, para ello utilizaremos la instrucción *const*, tal como vemos en este ejemplo:

```
const int maximo = 12345678;
```

Como podemos comprobar, tenemos que utilizar una instrucción para indicar que es una constante, (*const*), seguida del tipo de datos y el valor que le asignaremos a esa constante.

Declarar variables

En la declaración de las variables en Visual C# 2005, siempre hay que indicar el tipo de datos que tendrá la variable, por la sencilla razón de que las variables definidas en C# siempre se harán usando el tipo de datos sin necesidad de utilizar ninguna instrucción especial que indique que es una declaración.

Por ejemplo, en este ejemplo, estamos declarando una variable de tipo *int* y otra de tipo *decimal*:

```
int i;  
decimal m;
```

También podemos declarar más de una variable en la misma sentencia. Lo único que tendremos que hacer es separar las variables con comas. Por ejemplo, el siguiente código definimos tres variables del tipo *int*:



```
int a, b, c;
```

Declarar variables y asignar el valor inicial

Cuando declaramos variables en C#, estas estarán en un estado "no iniciado", es decir, no tendrán ningún valor inicial asignado de forma automática, salvo cuando estas variables están definidas como campos de una clase.

Por tanto, si queremos que en C# tengan un valor inicial, tendremos que asignarlos de forma expresa.

Por ejemplo:

```
int a = 10;
```

En esa misma línea podemos declarar y asignar más variables, simplemente separándolas con comas:

```
int b = 12, c = 15;
```

Si queremos declarar más de una variable pero con tipos diferentes tendremos que separarlas con un punto y coma, que es la forma de decirle al compilador que es una nueva instrucción, por tanto esta declaración:

```
decimal m = 22.5M; double d = 55.556;
```

Será lo mismo que esta otra:

```
decimal m = 22.5M;  
double d = 55.556;
```

Como es natural, a una variable podemos asignarle el contenido de una constante "con nombre" que previamente hayamos declarado, esa asignación también podemos hacerla al declarar la variable:

```
const int maximo = 12345678;  
int i = maximo;
```

El tipo de datos char

Tal como comentamos anteriormente, el tipo *char* de C# es un caso especial, realmente es un carácter Unicode, pero internamente es un valor *int*. Veamos primero cómo declarar variables (o constantes) de tipo *char* y después veremos que realmente son enteros (o al menos que en algunas circunstancias se comportan como tales):

```
char c1 = 'a';  
char c2 = (char)49;
```

En la primera declaración utilizamos una constante literal para asignarla a una variable de tipo *char*. Esa constante la indicaremos entre comillas simples.

En la segunda usamos una conversión explícita de un tipo numérico entero a un valor *char*.

Al ser un valor entero, podremos hacer operaciones aritméticas con variables de ese tipo, aunque el resultado siempre será un valor entero, no un carácter. En este código sumamos el valor de las dos constantes y lo asignamos en una variable de tipo *int*:

```
int i2 = c1 + c2;
```

Si queremos volver a convertir el contenido de la variable *i* en un valor de tipo *char* tendremos que hacer una conversión (*cast*) o bien usar la clase *Convert*.

```
char c3 = (char)i2;  
char c4 = Convert.ToChar(i2);
```

Cadenas de caracteres

Las cadenas de caracteres en .NET, y por tanto en C#, se definen indicando la cadena entre comillas dobles. Tal como vimos en la tabla 2.1, la instrucción para declarar cadenas es string:

```
string s1 = "Hola, ";
```

Tal como indicamos en el comentario de esa tabla, las cadenas de C# son inmutables, es decir una vez que hemos creado una cadena y la modificamos, se creará una nueva cadena y se desechará la cadena anterior.

Por ejemplo, si hacemos esto:

```
s1 = s1 + "mundo";
```

Se creará una nueva cadena en la que asignará el contenido de las dos usadas en la expresión de la derecha de la asignación, el contenido anterior de la variable s1 se desechará y se usará el resultado de unir lo que hubiera antes en esa variable más la cadena que indicamos después del signo igual.

Para paliar este pobre rendimiento de las cadenas en .NET, podemos usar la clase StringBuilder, la cual deberíamos usar preferentemente cuando realicemos concatenaciones o uniones de cadenas, ya que esa clase no crea nuevas instancias y para ello se utilizan métodos que nos permiten realizar esa unión de cadenas; el ejemplo anterior lo podemos convertir de esta forma:

```
StringBuilder sb = new StringBuilder();  
sb.Append("Hola, ");  
sb.Append("mundo");
```

2.3.- Sentencias Condicionales y Repetitivas

Sentencia if-else

La instrucción if selecciona una instrucción para ejecución en base al valor de una expresión Boolean. En el ejemplo siguiente un indicador Boolean flagCheck se establece en true y, a continuación, se protege en la instrucción if. El resultado es: The flag is set to true.

```
bool flagCheck = true;  
if (flagCheck == true)  
{  
    Console.WriteLine("The flag is set to true.");  
}  
else  
{  
    Console.WriteLine("The flag is set to false.");  
}
```

Si se desea ejecutar más de una instrucción, es posible ejecutar varias instrucciones en forma condicional al incluirlas en bloques mediante {}, al igual que en el ejemplo anterior.

Las instrucciones que se van a ejecutar como resultado de comprobar la condición pueden ser de cualquier tipo, incluida otra instrucción if anidada dentro de la instrucción if original. En las instrucciones if anidadas, la cláusula else pertenece a la última instrucción if que no tiene una cláusula else correspondiente. Por ejemplo:

```
if (x > 10)  
{  
    if (y > 20)  
        Console.WriteLine("Sentencia 1");  
}  
else  
    Console.WriteLine("Sentencia 2");
```

Sentencia switch



La instrucción switch es una instrucción de control que controla múltiples selecciones y enumeraciones pasando el control a una de las instrucciones case de su cuerpo, como se muestra en el ejemplo siguiente:

```
int caseSwitch = 1;
switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

El control se transfiere a la instrucción case que coincide con el valor del modificador. La instrucción switch puede incluir cualquier número de instancias case, sin embargo dos instrucciones case nunca pueden tener el mismo valor. La ejecución del cuerpo de la instrucción empieza en la instrucción seleccionada y continúa hasta que la instrucción break transfiere el control fuera del cuerpo case. Es necesario introducir una instrucción de salto como break después de cada bloque case, incluido el último bloque, se trate de una instrucción case o de una instrucción default. Con una excepción, (a diferencia de la instrucción switch de C++), C# no admite el paso implícito de una etiqueta case a otra. Esta excepción se produce si una instrucción case no tiene ningún código.

Sentencia for

El bucle for ejecuta una instrucción o un bloque de instrucciones repetidamente hasta que una determinada expresión se evalúa como false. El bucle for es útil para recorrer en iteración matrices y para procesar secuencialmente. En el ejemplo siguiente el valor de int i se escribe en la consola y el valor de i se va incrementando en 1 en el bucle.

```
using System;
class ForLoopTest
{
    static void Main()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

La instrucción for ejecuta la instrucción o instrucciones internas repetidamente del siguiente modo:

- Primero, se evalúa el valor inicial de la variable i.
- A continuación, mientras el valor de i sea menor o igual que 5, la condición se evalúa como true, se ejecuta la instrucción Console.WriteLine y se vuelve a evaluar i.
- Cuando i es mayor que 5, la condición se convierte en false y el control se transfiere fuera del bucle.

Puesto que la comprobación de la expresión condicional tiene lugar antes de la ejecución del bucle, las instrucciones internas de un bucle for pueden no llegar a ejecutarse.

Sentencia while

La sentencia while es un bucle condicional que se repite mientras la condición es verdadera. El bucle while nunca puede iterar si la condición comprobada inicialmente es falsa.

```
int i = 1;
while ( i <= 100)
{
    Console.WriteLine( i );
    i++;
}
```

El ejemplo anterior imprime en pantalla los cien primeros números naturales. Si se desea ejecutar más de una instrucción, es posible ejecutar varias instrucciones en forma iterativa al incluirlas en bloques mediante {}, al igual que en el ejemplo anterior.

Sentencia do

La sentencia do actúa como la sentencia while. La única diferencia real es que la evaluación y la prueba de salida del bucle se hace después que el cuerpo del bucle se ha ejecutado, en lugar de antes.

```
int i = 1;
do
{
    Console.WriteLine ( " El cuadrado de : " + i + " es : " + i * i++);
} while( i <= 100)
```

El ejemplo anterior visualiza el cuadrado de los cien primeros números naturales. En las líneas de código podemos apreciar que en primera instancia se ejecutan las sentencias que van a continuación de la expresión do y a continuación se evalúa la expresión condicional.

2.4.- Arrays (matrices)

Los arrays (o matrices) nos permitirán agrupar valores que de alguna forma queremos que estén relacionados entre sí.

Declarar arrays

En C# los arrays se definen indicando un par de corchetes en el tipo de datos.

Por ejemplo, para indicar que queremos tener un array llamado numeros para almacenar valores de tipo int, lo haremos de esta forma:

```
int[] numeros;
```

Esta declaración simplemente indica que la variable **numeros** "será" un array, pero aún no está inicializada.

Para iniciarla, al menos con valores cero, podemos hacerlo de estas dos formas, dependiendo de que ya hayamos declarado previamente la variable, (primera línea), o lo queramos hacer al declararla, (segunda línea):

```
numeros = new int[4];
int[] num2 = new int[3];
```

En ambos casos debemos usar new seguido del tipo de datos y entre corchetes el número de elementos que tendrá el array. Ese número indicará el total de elementos que tendrá el array, pero debido a como .NET trata los arrays, el índice inferior será cero y el índice superior será uno menos del número de elementos que hemos indicado al crear el array.



Para asignar valores a los elementos de un array, lo haremos como con cualquier variable, pero usando los corchetes y la posición en la que queremos guardar ese valor:

```
num2[0] = 3;
num2[1] = 22;
```

Para recorrer todos los elementos que tenga un array podemos hacerlo de la forma tradicional, es decir, usando un bucle for con una variable que recorra todos los índices, en cuyo caso necesitamos averiguar el valor del índice superior del array, cosa que haremos por medio de la propiedad Length, tal como podemos ver en este código:

```
for (int i = 0; i < num2.Length; i++)
    Console.WriteLine(num2[i]);
```

Como vemos, para acceder a cada uno de los valores, el elemento al que queremos acceder lo indicamos dentro de corchetes.

La otra forma de hacerlo es mediante un bucle foreach, pero debido a que foreach obtiene cada uno de los valores que le indiquemos, no necesitamos acceder directamente al array, sino que usaremos el valor obtenido en cada ciclo del bucle, tal como podemos comprobar en este trozo de código:

```
foreach(int i in num2)
    Console.WriteLine(i);
```

Declarar e inicializar un array

Al igual que podemos declarar variables e inicializarlas al mismo tiempo, con los arrays también podemos hacerlo, aunque la sintaxis es un poco distinta, ya que en esta ocasión debemos indicar varios valores. Esos valores los indicaremos justo después de definir el array:

```
string[] nombres = {"Pepe", "Juan", "Luisa"};
```

En este caso, cuando iniciamos el array al declararlo, no debemos indicar el número de elementos que tendrá ese array, ya que ese valor lo averiguará el compilador cuando haga la asignación. Tampoco es válido indicar el número de elementos que queremos que tenga y solo asignarle unos cuantos menos (o más), ya que se producirá un error en tiempo de compilación.

Si el array es bidimensional (o con más dimensiones), también podemos inicializarlos al declararlo, pero en este caso debemos usar doble juego de llaves:

```
string[,] nombres = { { "Juan", "Pepe" }, { "Ana", "Eva" } };
```

En este código tendríamos un array bidimensional con los siguientes valores:

```
nombres[0,0]= Juan
nombres[0,1]= Pepe
nombres[1,0]= Ana
nombres[1,1]= Eva
```

Como podemos ver en la declaración anterior, si definimos arrays con más de una dimensión, debemos indicarla, en la declaración del tipo, usando una coma para separar cada dimensión, o lo que es más fácil de recordar: usando una coma menos del número de dimensiones que tendrá el array. En los valores a asignar, usaremos las llaves encerradas en otras llaves, según el número de dimensiones.

Aunque, la verdad, es que hay algunas veces que hay que hacer un gran esfuerzo mental para asociar los elementos con los índices que tendrán en el array, por tanto, algunas veces puede que resulte más legible si indentamos o agrupamos esas asignaciones, tal como vemos en el siguiente código:

```
string[,] nomTri =
{
    { { "Juan", "Pepe" }, { "Luisa", "Eva" } },
    { { "A", "B" }, { "C", "D" } }
};
```

```

Console.WriteLine(nomTri[0, 0, 0]); // Juan
Console.WriteLine(nomTri[0, 0, 1]); // Pepe
Console.WriteLine(nomTri[0, 1, 0]); // Luisa
Console.WriteLine(nomTri[0, 1, 1]); // Eva
Console.WriteLine(nomTri[1, 0, 0]); // A
Console.WriteLine(nomTri[1, 0, 1]); // B
Console.WriteLine(nomTri[1, 1, 0]); // C
Console.WriteLine(nomTri[1, 1, 1]); // D

```

Cambiar el tamaño de un array

Visual C# 2005 no nos ofrece una forma fácil de cambiar el tamaño de un array, de hecho no tiene ninguna instrucción que sirva para eso.

Aunque redimensionar un array realmente no tiene ningún problema, ya que simplemente lo haremos de la misma forma que vimos anteriormente:

```
num2 = new int[4];
```

E incluso podemos asignarle nuevos valores al "redimensionarlo", usando la misma sintaxis que vimos en la sección anterior:

```
num2 = new int[] { 6, 7, 8, 9 };
```

Lo que no podemos hacer en C# es cambiar el tamaño de un array conservando el contenido anterior, algo que en otros lenguajes si es posible. La única forma de conseguirlo es hacer una copia previa de ese array, redimensionarlo al nuevo tamaño y volver a copiar el contenido anterior.

En el siguiente código podemos ver estos pasos:

```

int[] original = { 1, 2, 3, 4 };
int[] copia = new int[original.Length];
// Copiamos el original en la copia
original.CopyTo(copia, 0);
// Añadimos 5 elementos más
original = new int[original.Length + 5];
// Asignamos nuevamente lo copiado a partir del primer elemento
copia.CopyTo(original, 0);

```

Para hacer la copia hemos usado el método CopyTo, al que le indicamos el array de destino, (que previamente debe estar dimensionado con el número de elementos que recibirá), y el índice a partir del cual queremos copiar ese array, en nuestro caso le indicamos la primera posición, que como sabemos es la posición cero.

Si el array contiene valores por valor, podemos usar otro método: Clone, ya que este método devuelve una copia "superficial" del objeto en cuestión, en nuestro caso el array; esa copia superficial significa que solo copiará el contenido del array, si este tuviera referencia a objetos, no copiaría los objetos, solo las referencias, (de todo esto de los objetos y referencias a objetos nos ocuparemos en la próxima lección), pero el valor que devuelve ese método es del tipo object, es decir el tipo genérico de .NET, por tanto tendríamos que hacer una conversión al tipo array, tal como vemos en este código:

```
int[] copia = (int[])original.Clone();
```

Eliminar el contenido de un array

Una vez que hemos declarado un array y le hemos asignado valores, es posible que nos interese eliminar esos valores de la memoria, para lograrlo, en C# podemos hacerlo de dos formas:

1. Redimensionando el array indicando que tiene cero elementos, aunque en el mejor de los casos, si no estamos trabajando con arrays de más de una dimensión, tendríamos



un array de un elemento, ya que, como hemos comentado anteriormente, los arrays de .NET el índice inferior es cero.

2. Asignar un valor *null* al array. Esto funciona en C# porque los arrays son tipos por referencia.

Los arrays son tipos por referencia

Como acabamos de ver, en Visual C# 2005 los arrays son tipos por referencia, y tal como comentamos anteriormente, los tipos por referencia realmente lo que contienen son una referencia a los datos reales no los datos propiamente dichos.

¿Cuál es el problema?

Veámoslo con un ejemplo y así lo tendremos más claro.

```
string[] nombres = { "Juan", "Pepe", "Ana", "Eva" };  
string[] otros;  
otros = nombres;  
nombres[0] = "Antonio";  
Console.WriteLine(otros[0]);
```

En este ejemplo definimos el array `nombres` y le asignamos cuatro valores. A continuación definimos otro array llamado `otros` y le asignamos lo que tiene `nombres`. Por último asignamos un nuevo valor al elemento cero del array `nombres`.

Si mostramos el contenido de ambos arrays nos daremos cuenta de que realmente solo existe una copia de los datos en la memoria, y tanto `nombres[0]` como `otros[0]` contienen el nombre "Antonio".

¿Qué ha ocurrido?

Debido a que los arrays son tipos por referencia, solamente existe una copia de los datos en la memoria y tanto la variable `nombres` como la variable `otros` lo que contienen es una referencia (o puntero) a los datos.

Si realmente queremos tener copias independientes, debemos hacer una copia del array utilizando el método `CopyTo` que ya vimos anteriormente.

Además de los métodos que hemos visto y de la propiedad `Length` que nos sirve para averiguar cuantos elementos tiene un array, tenemos más métodos y propiedades que podemos usar, por ejemplo para saber cuantas dimensiones tiene un array (`Rank`) o saber cual es el índice superior de una dimensión determinada (`GetUpperBound`).

Para finalizar este tema, sólo nos queda por decir, que los arrays de .NET, y por tanto los de C#, realmente son tipos de datos derivados de la clase `Array` y por tanto disponen de todos los miembros definidos en esa clase, aunque de esto hablaremos en la próxima lección, en la que también tendremos la oportunidad de profundizar un poco más en los tipos por referencia y en cómo podemos definir nuestros propios tipos de datos, tanto por referencia como por valor.

2.5.- Manejo de excepciones

En Visual C# 2005 se utiliza un tratamiento estructurado de excepciones, de esta forma podemos detectar los errores que se produzcan en nuestras aplicaciones de una forma más "ordenada".

Manejo de excepciones estructuradas

Las excepciones en C# las podemos controlar usando las instrucciones *try / catch / finally*. Estas instrucciones realmente son bloques de instrucciones, y por tanto estarán delimitadas con un par de llaves.

Cuando queramos controlar una parte del código que puede producir un error lo incluimos dentro del bloque *try*, si se produce un error, éste lo podemos detectar en el bloque *catch*, por último, independientemente de que se produzca o no una excepción, podemos ejecutar el código que incluyamos en el bloque *finally*.

Cuando creamos una estructura de control de excepciones no estamos obligados a usar los tres bloques, aunque el primero: *try* si es necesario, ya que es el que le indica al compilador que tenemos intención de controlar los errores que se produzcan. Por tanto podemos crear un "manejador" de excepciones usando los tres bloques, usando *try* y *catch* o usando *try* y *finally*.

Veamos ahora con más detalle cada uno de estos bloques y que es lo que podemos hacer en cada uno de ellos.

Bloque try

En este bloque incluiremos el código en el que queremos comprobar los errores. El código a usar será un código normal, es decir, no tenemos que hacer nada en especial, ya que en el momento que se produzca el error se usará (si hay) el código del bloque *catch*.

Bloque catch

Si se produce una excepción, ésta la capturamos en un bloque *catch*.

En el bloque *catch* podemos indicar que tipo de excepción queremos capturar, para ello usaremos una variable de tipo *Exception*, la cual puede ser del tipo de error específico que queremos controlar o de un tipo genérico.

Por ejemplo, si sabemos que nuestro código puede producir un error al trabajar con ficheros, podemos usar un código como éste:

```
try
{
    // código para trabajar con ficheros, etc.
}
catch(System.IO.IOException ex)
{
    // el código a ejecutar cuando se produzca ese error
}
```

Si nuestra intención es capturar todos los errores que se produzcan, es decir, no queremos hacer un filtro con errores específicos, podemos usar la clase *Exception* como tipo de excepción a capturar. La clase *Exception* es la más genérica de todas las clases para manejo de excepciones, por tanto capturará todas las excepciones que se produzcan.

```
try
{
    // código que queremos controlar
}
catch(System.Exception ex)
{
    // el código a ejecutar cuando se produzca cualquier error
}
```

Aunque si no vamos usar la variable indicada en el bloque *Catch*, pero queremos que no se detenga la aplicación cuando se produzca un error, podemos hacerlo de esta forma:

```
try
{
    // código que queremos controlar
}
catch
{
}
```

```
    // el código a ejecutar cuando se produzca cualquier error  
}
```

La variable indicada en el bloque catch la podemos usar para mostrar un mensaje al usuario o para obtener información extra sobre el error, pero no siempre vamos a hacer uso de esa variable, en ese caso podemos utilizar el código anterior, en el que no se usa una variable y tampoco se indica el tipo de error que queremos interceptar.

Pero es posible que nuestra intención sea capturar errores de un tipo concreto sin necesidad de utilizar una variable, en ese caso podemos crear un bloque catch como el siguiente, en el que solo se indica el tipo de excepción:

```
try  
{  
    // código que queremos controlar  
}  
catch(FormatException)  
{  
    // interceptar los errores del tipo FormatException  
}
```

Varías capturas de errores en un mismo bloque try/catch

En un mismo try/catch podemos capturar diferentes tipos de errores, para ello podemos incluir varios bloques catch, cada uno de ellos con un tipo de excepción diferente.

Es importante tener en cuenta que cuando se produce un error y usamos varios bloques catch, el CLR de .NET buscará la captura que mejor se adapte al error que se ha producido, pero siempre lo hará examinando los diferentes bloques catch que hayamos indicado empezando por el indicado después del bloque try, por tanto deberíamos poner las más genéricas al final, de forma que siempre nos aseguremos de que las capturas de errores más específicas se intercepten antes que las genéricas.

Aunque el propio compilador de C# detectará si hay capturas de errores genéricas antes que las más específicas, avisándonos de ese hecho. En el siguiente código capturamos un error específico y también uno genérico, con idea de que tengamos siempre controlado cualquier error que se produzca:

```
try  
{  
    // código que queremos controlar  
}  
catch(FormatException)  
{  
    // captura de error de formato  
}  
catch(Exception ex)  
{  
    // captura del resto de errores  
}
```

Bloque finally

En este bloque podemos indicar las instrucciones que queremos que se ejecuten, se produzca o no una excepción. De esta forma nos aseguramos de que siempre se ejecutará un código, por ejemplo para liberar recursos, se haya producido un error o no.

En este código tenemos tres capturas de errores diferentes y un bloque finally que siempre se ejecutará, se produzca o no un error:

```
int i, j;  
//  
try
```

```

{
    Console.Write("Un numero ");
    i = Convert.ToInt32(Console.ReadLine());
    Console.Write("Otro numero ");
    j = Convert.ToInt32(Console.ReadLine());
    int r = i / j;
    Console.WriteLine("El resultado es: {0}", r);
}
catch (FormatException)
{
    //
    Console.WriteLine("No es un número válido");
    // Salimos de la función, pero se ejecutará el finally
    return;
}
catch (DivideByZeroException)
{
    Console.WriteLine("La división por cero no está permitida.");
}
catch (Exception ex)
{
    //
    // Captura del resto de excepciones
    Console.WriteLine(ex.Message);
}
finally
{
    //
    // Este código siempre se ejecutará
    Console.WriteLine("Se acabó");
}

```

Captura de errores no controlados

Como es lógico, si no controlamos las excepciones que se puedan producir en nuestras aplicaciones, éstas serán inicialmente controladas por el propio runtime de .NET, en estos casos la aplicación se detiene y se muestra el error al usuario.

Pero esto es algo que no deberíamos consentir, por tanto siempre deberíamos detectar todos los errores que se produzcan en nuestras aplicaciones, pero a pesar de que lo intentemos, es muy probable que no siempre podamos conseguirlo.

Una forma de hacerlo es iniciando nuestra aplicación dentro de un bloque try/catch, de esta forma, cuando se produzca el error, se capturará en ese bloque catch, porque cuando el runtime de .NET se encuentra con una excepción, lo que hace es revisar "la pila" de llamadas y buscar algún try/catch, si lo encuentra, lo utiliza, y si no lo encuentra, se encarga de lanzar la excepción deteniendo el programa.

Esto es importante saberlo, no ya por detectar esos errores que no hemos tenido la previsión de controlar, sino porque es posible que si un error se produce dentro de un método en el que no hay captura de errores, pero antes de llamar a ese método hemos usado un try/catch, el error será interceptado por ese catch, aunque posiblemente ni siquiera lo pusimos pensando que podía capturar errores producidos en otros niveles más profundos de nuestra aplicación.

Estructuras: Tipos por valor definidos por el usuario

De la misma forma que podemos definir nuestros propios tipos de datos por referencia, Visual C# nos permite crear nuestros propios tipos por valor.

Para crear nuestros tipos de datos por referencia, usamos la "instrucción" class, por tanto es de esperar que también exista una instrucción para crear nuestros tipos por valor, y esa instrucción es: struct, de ahí que los tipos por valor definidos por el usuario se llamen estructuras.

Las estructuras pueden contener los mismos miembros que las clases, aunque algunos de ellos se comporten de forma diferente o al menos tengan algunas restricciones, como que los campos definidos en las estructuras no se pueden inicializar al mismo tiempo que se declaran o no pueden contener constructores "simples", ya que el propio compilador siempre se encarga de crearlo, para así poder inicializar todos los campos definidos.

Otra de las características de las estructuras es que no es necesario crear una instancia para poder usarlas, ya que es un tipo por valor y los tipos por valor no necesitan ser instanciados para que existan.

Definir una estructura

Las estructuras se definen usando la palabra `struct` seguida del nombre, en código de las estructuras, al igual que el de las clases o cualquier otro bloque de código de C# lo incluiremos entre un par de llaves.

El siguiente código define una estructura llamada `Punto` en la que tenemos dos campos públicos.

```
struct Punto
{
    public int X;
    public int Y;
}
```

Para usarla podemos hacer algo como esto:

```
Punto p;
p.X = 100;
p.Y = 75;
```

También podemos usar `new` al declarar el objeto:

```
Punto p = new Punto();
```

Aunque en las estructuras, usar `new`, sería algo redundante y por tanto no necesario.

Constructores de las estructuras

Tal y como hemos comentado, las estructuras siempre definen un constructor sin parámetros, este constructor no lo podemos definir nosotros, es decir, siempre existe y es el que el propio compilador de Visual C# define.

Por tanto, si queremos agregar algún constructor a una estructura, este debe tener parámetros y, tal como ocurre con cualquier método o como ocurre en las clases, podemos tener varias sobrecargas de constructores parametrizados en las estructuras. La forma de definir esos constructores es como vimos en las clases: usando distintas sobrecargas de un "método" llamado de la misma forma que la estructura. En el caso de las estructuras también podemos usar la palabra clave `this` para referirnos a la instancia actual.

Esto es particularmente práctico cuando queremos dejar claro de que estamos llamando a un método de la instancia actual.

Destruidores de las estructuras

Debido a que las estructuras son tipos por valor y por tanto una variable declarada con un tipo por valor "contiene el valor en sí misma", no podemos destruir este tipo de datos, lo más que conseguiríamos al asignarle un valor nulo (`null`) sería eliminar el contenido de la variable, pero

nunca podemos destruir ese valor. Por tanto, en las estructuras no podemos definir destructores.

Los miembros de una estructura

Como hemos comentado, los miembros o elementos que podemos definir en una estructura son los mismos que ya vimos en las clases. Por tanto aquí veremos las diferencias que existen al usarlos en las estructuras.

Campos

Como vimos, las variables declaradas a nivel del tipo, son los campos, la principal diferencia con respecto a las clases, es que los campos de una estructura no pueden inicializarse en la declaración y el valor que tendrán inicialmente es un valor "nulo", que en el caso de los campos de tipo numéricos es un cero.

Por tanto, si necesitamos que los campos tengan algún valor inicial antes de usarlos, deberíamos indicárselo a los usuarios de nuestra estructura y proveer un constructor que realice las inicializaciones correspondientes, pero debemos recordar que ese constructor debe tener algún parámetro, ya que el predeterminado sin parámetros no podemos "reescribirlo".

Los únicos campos que podemos inicializar al declararlos son los campos compartidos, pero como tendremos oportunidad de ver, estos campos serán accesibles por cualquier variable declarada y cualquier cambio que realicemos en ellos se verá reflejado en el resto de "instancias" de nuestro tipo.

Métodos y otros elementos

El resto de los miembros de una estructura se declaran y usan de la misma forma que en las clases. De los métodos podemos crear sobrecargas, definirlos compartidos (estáticos) o de instancia, no existiendo ningún tipo de restricción en este aspecto, ya que podemos definir una estructura sin métodos o con todos los métodos compartidos, lo mismo es aplicable a las propiedades, salvo que éstas no pueden tener sobrecarga.

Cómo usar las estructuras

Tal como hemos comentado las estructuras son tipos por valor, para usar los tipos por valor no es necesario instanciarlos explícitamente, ya que el mero hecho de declararlos indica que estamos creando un nuevo objeto en memoria. Por tanto, a diferencia de las clases o tipos por referencia, cada variable definida como un tipo de estructura será independiente de otras variables declaradas, aunque no las hayamos instanciado.

Esta característica de las estructuras nos permite hacer copias "reales", no una copia de la referencia (o puntero) al objeto en memoria, como ocurre con los tipos por referencia. Veámoslos con un ejemplo:

```
Punto p = new Punto(100, 75);
Punto p1;
p1 = p;
p1.X = 200;
// p.X vale 100 y p1.X vale 200
```

En este trozo de código definimos e "instanciamos" una variable del tipo Punto, a continuación declaramos otra variable del mismo tipo y le asignamos la primera; si estos tipos fuesen por referencia, tanto una como la otra estarían haciendo referencia al mismo objeto en memoria, y cualquier cambio realizado a cualquiera de las dos variables afectarían al mismo objeto, pero en el caso de las estructuras (y de los tipos por valor), cada cambio que realicemos se hará

sobre un "objeto" diferente, por tanto la asignación del valor 200 al campo X de la variable p1 solo afecta a esa variable, dejando intacto el valor original de la variable p.



RESUMEN

Visual C# 2005 es un lenguaje de programación muy apropiado para construir sistemas de información basados en red o mejor aún en Internet. C# es una evolución de C++ en el que han influido también ideas propias de otros lenguajes. Lo más importante es que se trata de un lenguaje orientado a componentes, el lenguaje en el que se han escrito los servicios de la plataforma .NET, un lenguaje flexible, potente, claro y elegante.



BIBLIOGRAFÍA RECOMENDADA

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

Fundamentos de Programación – *Luis Joyanes* – Mc Graw Hill.



NEXO

La unidad TEMÁTICA tratada nos brindó un panorama ampliado de los fundamentos de programación con el lenguaje C#.



ACTIVIDAD

Implemente programas para cada uno de los temas tratados que contienen código de ejemplo en la unidad TEMÁTICA. Realice un análisis exhaustivo de los mismos.



AUTOEVALUACIÓN FORMATIVA

Desarrolla los siguientes problemas utilizando Microsoft Visual C#:

1.- Escribir un programa que permita eliminar todas las ocurrencias de cada carácter en una cadena dada a partir de otra cadena dada. Las dos cadenas son:

CADENA1 Es la cadena donde deben eliminarse caracteres.

LISTA Es la cadena que proporciona los caracteres que deben eliminarse.

CADENA = "EL EZNZZXTX"

LISTA = "XZ"

Resultado = "EL ENT"

- 2.- Una empresa tiene diez almacenes y necesita crear un programa que lea las ventas mensuales de los diez almacenes, calcule la media de ventas y obtenga el listado de los almacenes cuyas ventas mensuales son superiores a la media. Utilice arrays.

Unidad Temática III

DESARROLLO DE APLICACIONES WINDOWS

3.1.- Uso del diseñador de visual Studio 2005

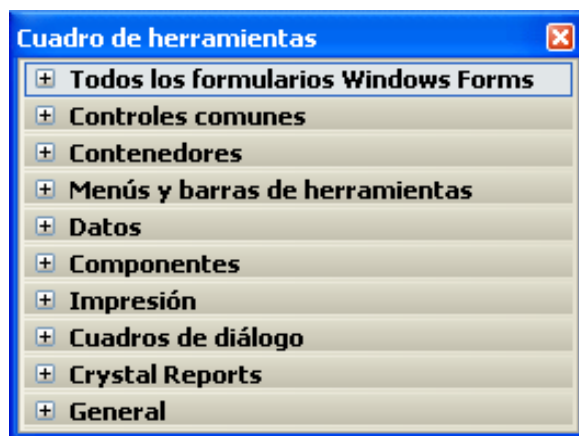
Cuando nos encontramos con Visual Studio 2005 por primera vez, saltan a la vista, algunos de los cambios más importantes de este novedoso entorno de desarrollo de aplicaciones Windows.

Para un desarrollador, familiarizarse con el entorno de Visual Studio 2005 es una tarea que no debe entrañar una complejidad excesivamente grande. Como nos ocurre a todos los que nos encontramos delante de un nuevo entorno de trabajo, lo único que se requiere es constancia y práctica, mucha práctica. Sin embargo, si usted es ya un desarrollador habitual de otros entornos de desarrollo, notará que sus avances van a ser significativos en muy poco tiempo.

Cuadro de herramientas

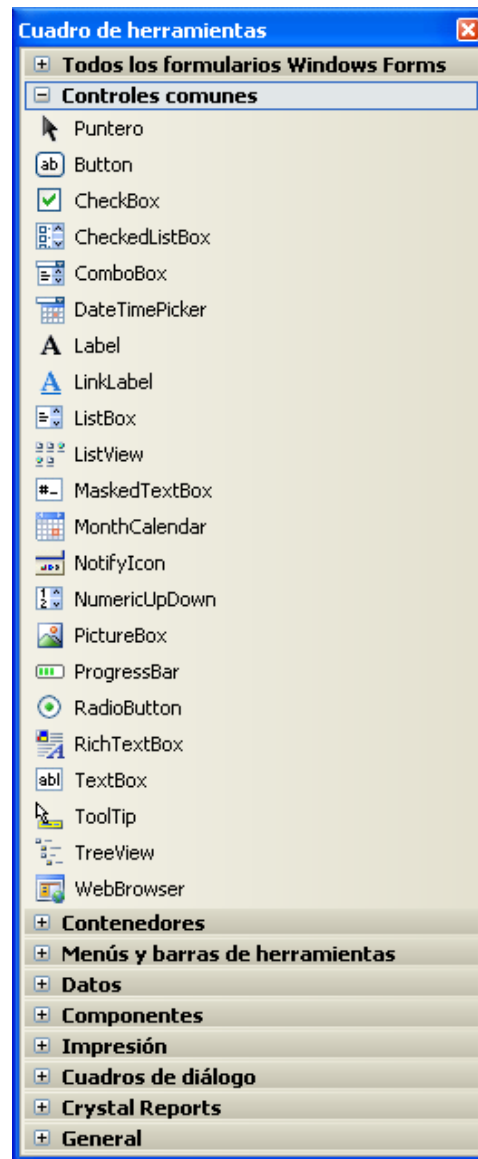
El cuadro o barra de herramientas de Visual Studio 2005, nos permite utilizar los distintos componentes que .NET Framework pone a nuestra disposición, en Visual Studio 2005 tenemos una gran cantidad de controles dispuestos en diferentes categorías.

En la siguiente figura puede ver la barra de herramientas de Visual Studio 2005.



El Cuadro de herramientas, lo localizará en la parte izquierda del entorno Visual Studio 2005.

Cuando iniciamos un nuevo proyecto con Visual Studio 2005, el cuadro de herramientas queda relleno con los controles que podemos utilizar en el proyecto. Si abrimos un formulario Windows, los controles quedan habilitados para que los podamos insertar en el formulario Windows. En la siguiente figura se muestra la barra de herramientas con los controles preparados para ser insertados en el formulario Windows.



Toolbox de Visual Studio 2005 con controles Windows preparados para ser insertados en el formulario Windows

Para insertar un control en un formulario Windows, se requiere que el formulario Windows sobre el que deseamos insertar un control, esté abierto. Una vez que está abierto, bastará con realizar una de las tres siguientes acciones para insertar un control al formulario:

- Hacer doble clic sobre un control del cuadro de herramientas.
- Hacer clic sobre un control del cuadro de herramientas, y sin soltar el botón del mouse, arrastrarlo sobre el formulario.
- Hacer clic sobre un control del cuadro de herramientas, y luego hacer clic sobre el formulario y arrastrar para marcar una zona que cubrirá nuestro control y soltar el mouse.

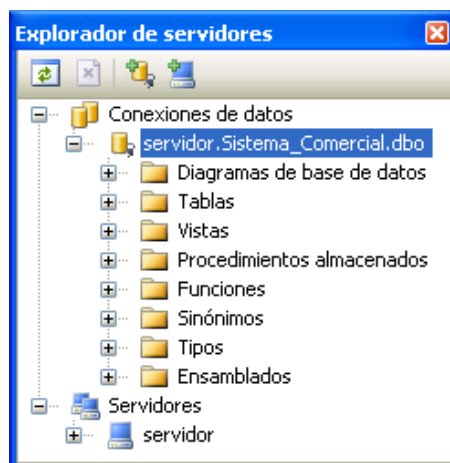
El control quedará entonces insertado dentro del formulario.

3.2.- Explorador de base de datos

Si ha sido lo suficientemente observador cuando se explicaban los detalles del cuadro o barra de herramientas, y ha prestado especial atención a las figuras o a las ventanas del entorno de desarrollo de Visual Studio 2005 Express, quizás haya notado que en la parte izquierda además de la solapa *cuadro de herramientas*, aparece otra solapa de nombre *explorador de base de datos*.

Desde esta solapa, un programador puede acceder a diferentes recursos del sistema. El principal y más importante recurso, es el que tiene que ver con las conexiones con bases de datos, ya sean Microsoft Access, Microsoft SQL Server o cualquier otra fuente de datos.


En la siguiente figura puede observar la solapa *Explorador de base de datos* extendida con parte de sus opciones.

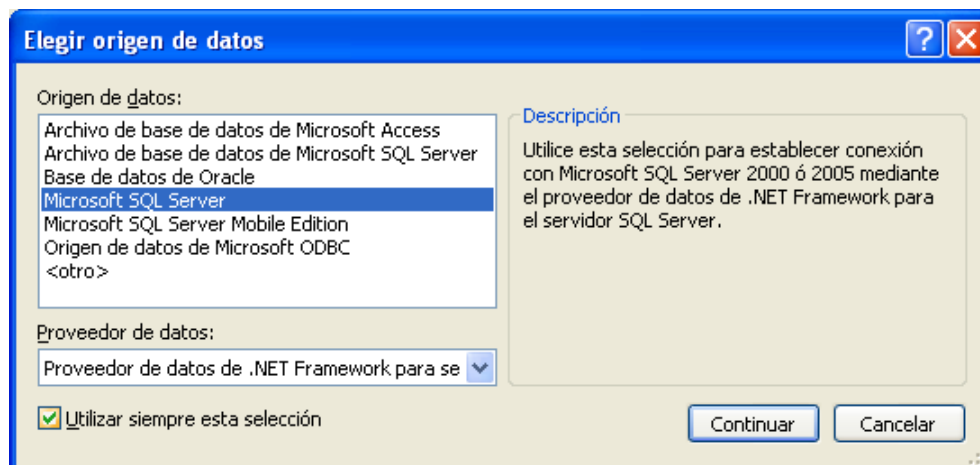


La solapa del Explorador de base de datos desplegada de Visual Studio 2005

Conexión a una base de datos Microsoft SQL Server 2005 a través de OLE DB

Para muestra un botón, y dado el carácter práctico de este texto, aprenderá a crear una conexión con cualquier base de datos, en nuestro caso de ejemplo una base de datos Microsoft SQL Server 2005, para poder utilizarla fácilmente en una aplicación Windows.

Haga clic sobre el botón representado por la siguiente imagen . En este instante, se abrirá una nueva ventana.



Ventana Elegir origen de datos

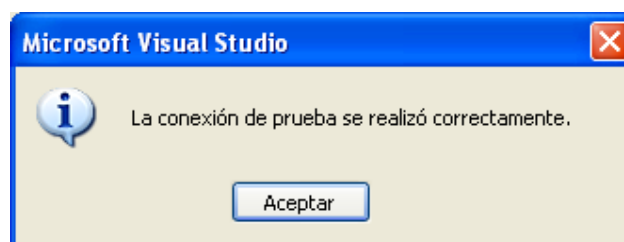
Si deseamos realizar una conexión a un servidor de base de datos como lo es SQL Server 2005 debemos seleccionar en la sección origen de datos, la opción Microsoft SQL Server como muestra la imagen, luego deberemos hacer clic sobre el botón Continuar y se visualizará la siguiente ventana.

Ventana agregar conexión

A continuación debemos ingresar o seleccionar un Nombre de Servidor, el que será quién provea de los datos para nuestras aplicaciones. Debemos también seleccionar el tipo de autenticación, para nuestro caso seleccionamos Utilizar autenticación Windows. En la sección establecer conexión con una base de datos, debemos seleccionar la base de datos que nos servirá de origen de datos.

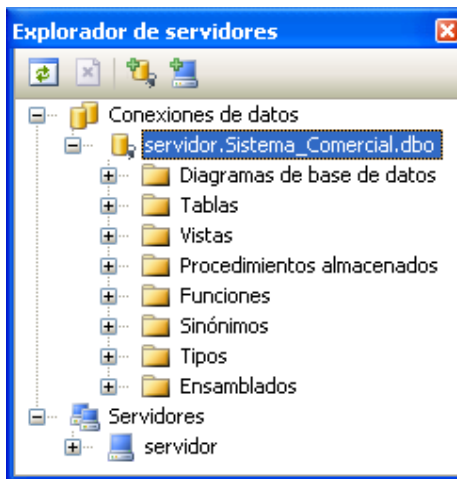
Ventana agregar conexión con parámetros seleccionados

Seguidamente debemos probar la conexión haciendo clic en el botón Probar conexión, de ser correcta visualizaremos la siguiente ventana.



Visualizamos la correcta conexión a una base de datos de SQL Server.

En este punto, tan sólo deberemos presionar el botón Aceptar para que la base de datos con la que hemos establecido la conexión, quede ahora insertada en la ventana del Explorador de base de datos como se muestra en la figura.



Base de datos Microsoft SQL Server 2005 insertada en la ventana del Explorador de base de datos

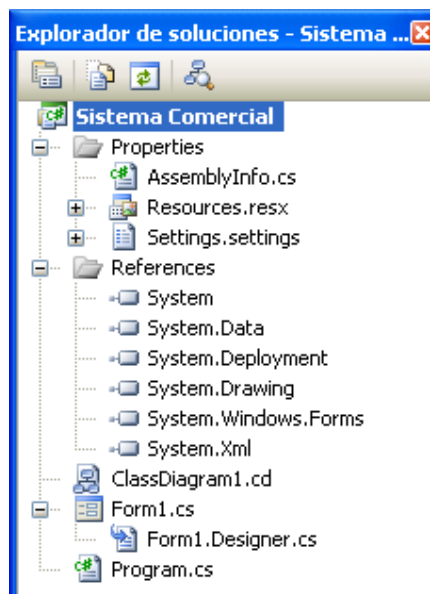
3.3.- Explorador de soluciones

El *Explorador de soluciones* lo podemos encontrar en la parte derecha de nuestro entorno de desarrollo.

Una solución se compone de proyectos y éstos, de recursos y objetos. Por lo general, una solución contendrá un proyecto, pero podemos encontrarnos con más de un proyecto dentro de una misma solución.

Sin embargo, estos conceptos son muy sencillos de comprender y controlar, y para nada debe hacernos pensar que esto es algo complejo que nos costará mucho tiempo dominar.

En la figura, podemos observar el explorador de soluciones de *Visual Studio 2005*.



La opción Explorador de soluciones desplegada en Visual Studio 2005

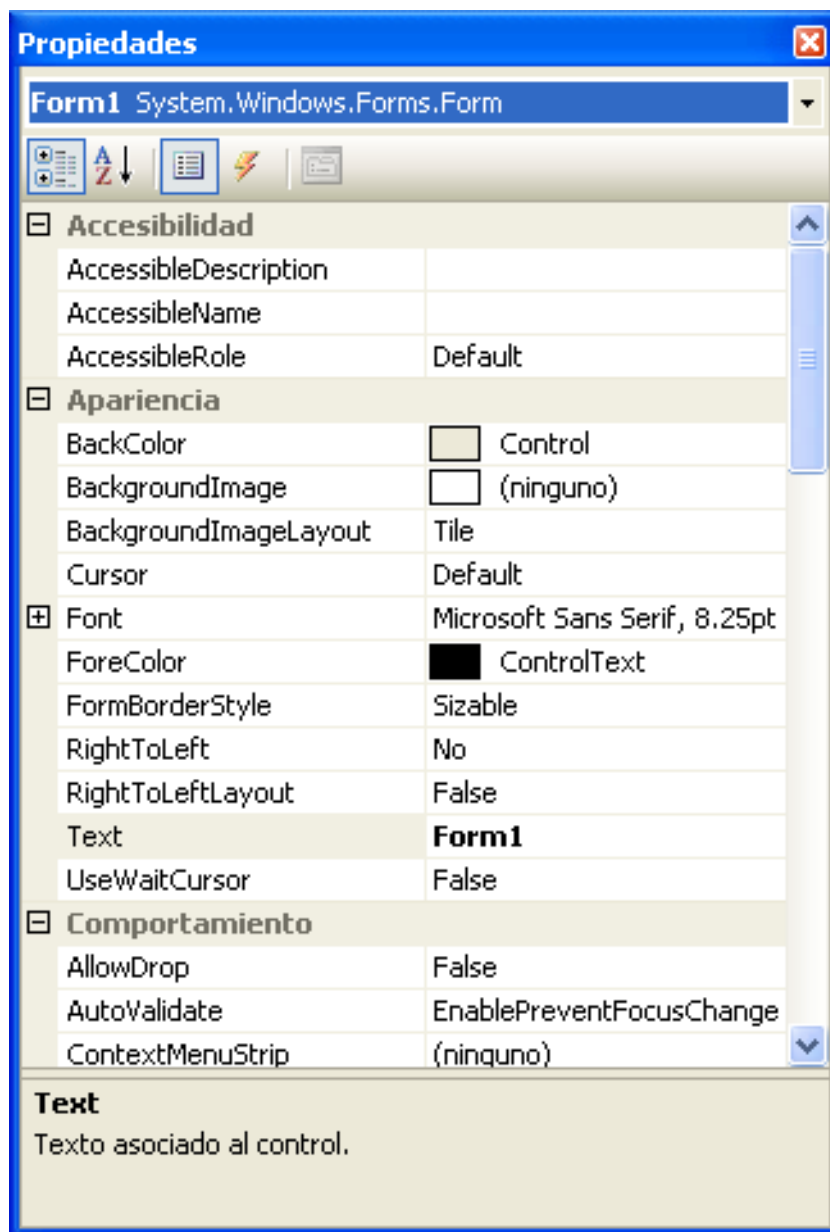
Si queremos añadir un nuevo formulario al proyecto, lo haremos presionando con el botón secundario en cualquier parte de la ventana del explorador de soluciones, pero si esa pulsación la hacemos en alguno de los objetos que contiene el proyecto, no podremos hacerlo, ya que el IDE de Visual Studio 2005 muestra un menú diferente según el objeto presionado, por ejemplo

si queremos añadir un nuevo proyecto, podemos hacerlo presionando con el botón secundario del mouse sobre la "solución".

3.4.- Propiedades

La ventana de propiedades la encontraremos en la parte derecha y más abajo de la ventana **Explorador de soluciones** en nuestro entorno de desarrollo.

Esta ventana nos permitirá acceder a las propiedades de los objetos insertados en nuestros formularios Windows, como se muestra en la figura.

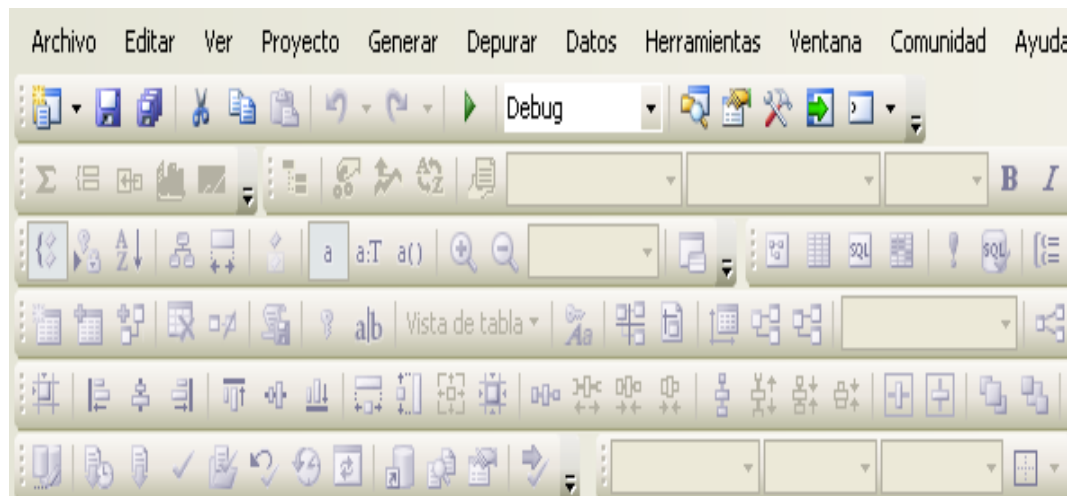


Ventana de Propiedades de Visual Studio 2005

Para acceder a las propiedades de un determinado control, deberemos seleccionar el control en el formulario Windows y acudir a la ventana **Propiedades**, o bien, seleccionar el control en el formulario Windows y presionar la tecla **F4**.

3.5.- Menús y barra de botones

Respecto a los menús y barra de botones, son muchas las opciones que tenemos disponibles, tal como podemos comprobar en la figura.



Los menús y barras de botones de Visual Studio 2005

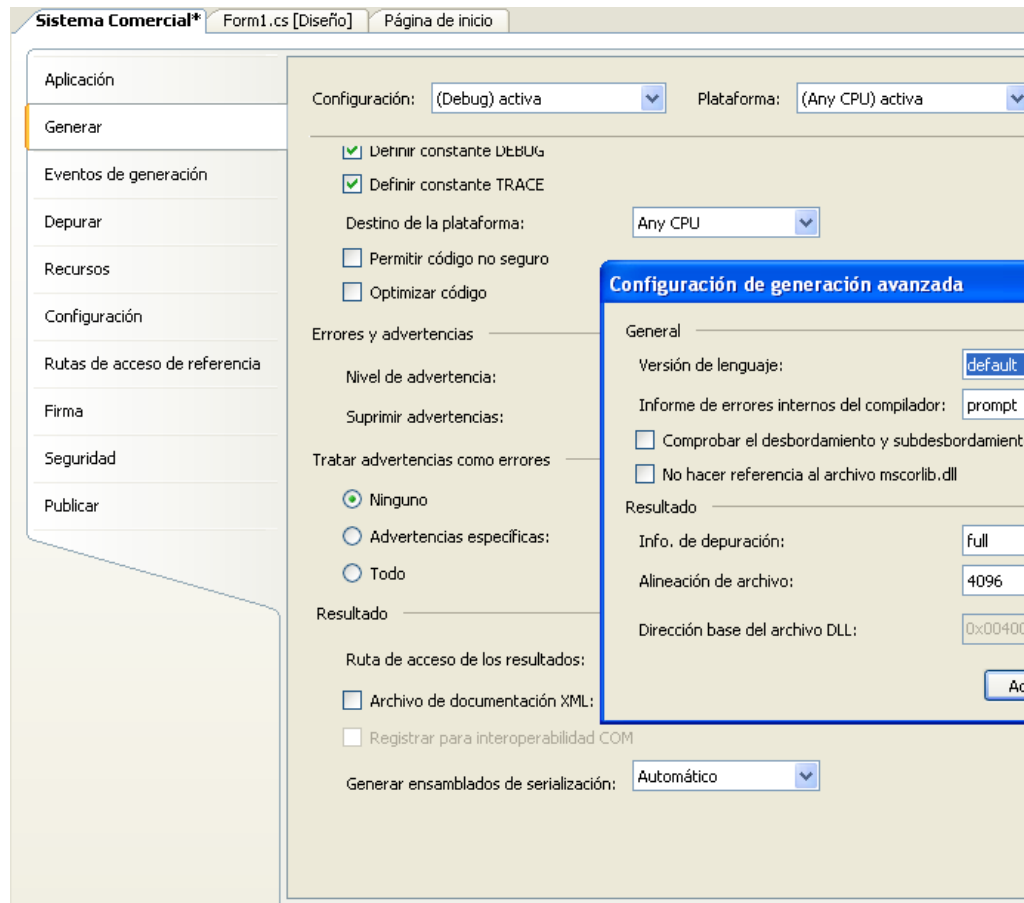
Las barras de botones son configurables, además de que podemos elegir las que queremos que se muestren de forma permanente en el entorno de desarrollo de Visual Studio 2005. Algunas de las barras de botones se mostrarán automáticamente según las tareas que estemos realizando, por ejemplo, cuando estamos en modo depuración o diseñando las tablas de una base de datos.

Con el contenido de los menús ocurre lo mismo, según el elemento que tengamos seleccionado se mostrarán ciertas opciones que sea relevantes para ese elemento del IDE de Visual Studio 2005.

Algunas de las opciones que tenemos en los menús también las podemos conseguir usando los menús contextuales (el mostrado al presionar con el botón secundario del mouse), y como es de esperar, también serán diferentes según el elemento que hemos presionado.

Por ejemplo, para configurar el proyecto actual, podemos elegir la opción **Propiedades** del menú **Proyecto** o bien presionar con el botón secundario del mouse sobre el proyecto mostrado en el Explorador de soluciones.

Al seleccionar las propiedades del proyecto, tendremos una nueva ventana desde la que podemos configurar algunas de las características de nuestro proyecto. En la siguiente figura, tenemos esa ventana de propiedades del proyecto, en la que podemos apreciar que está dividida según el tipo de configuración que queremos realizar, en este caso concreto las opciones de generación o compilación del proyecto.



Propiedades del proyecto sobre la que se trabaja en Visual Studio 2005

Como vemos en la figura, existen sin embargo multitud de opciones y apartados diferentes relacionados todos ellos con nuestra solución.

Otro de los apartados destacables, es el apartado denominado **Publicar**.

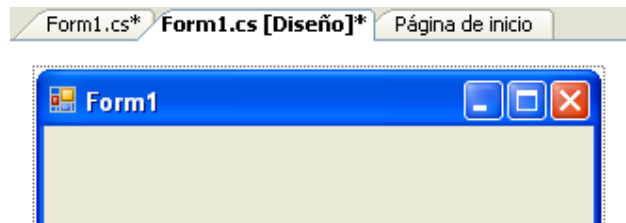
Aún así, éste es el corazón o parte fundamental que debemos controlar a la hora de desarrollar una aplicación o a la hora de gestionar una solución, porque dentro de esta ventana, se resume buena parte de los menús y barra de botones del entorno de Visual Studio 2005.

De todos los modos, tendremos la oportunidad de ver más adelante, algunos usos de algunas de las opciones de la barra de botones del entorno.

3.6.- Otras consideraciones

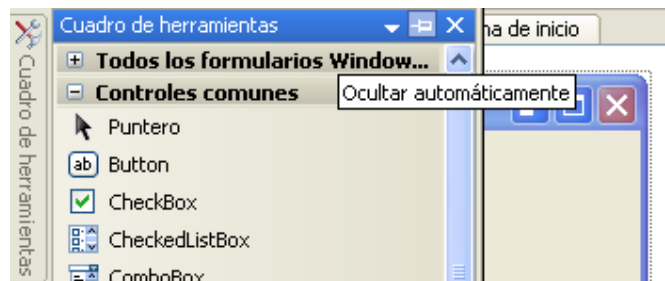
El desarrollador que haya utilizado previamente otros entornos de desarrollo distinto a los de la familia de Visual Studio .NET, encontrará muy interesantes algunos de los cambios incorporados en Visual Studio 2005. Al principio, quizás se encuentre un poco desorientado, pero rápidamente y gracias a su experiencia en otros entornos de desarrollo, se acostumbrará al cambio. Entre algunos de estos cambios, destacaría los siguientes:

- En Visual Studio 2005, acceder a los objetos de nuestra aplicación es mucho más fácil. Dentro del entorno, observaremos que se van creando diferentes solapas que nos permite acceder y localizar los recursos con los que estamos trabajando de forma rápida.



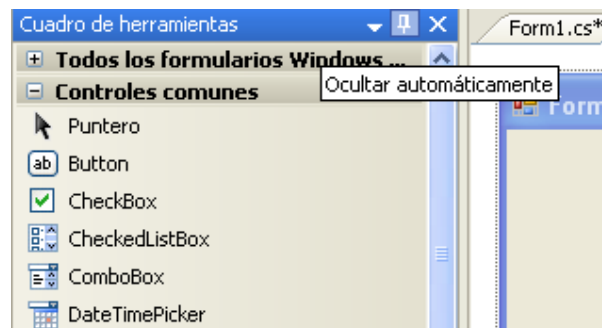
Solapas de los objetos abiertos en Visual Studio 2005

- Visual C# 2005 permite, hacer un Stop & Go (editar y continuar), de nuestras aplicaciones, es decir, pausar la ejecución de una aplicación en modo depuración y modificar los valores o propiedades que deseemos y continuar ejecutándola. Esta opción que los programadores de Visual Basic 6 utilizan con mucha frecuencia en el desarrollo de sus aplicaciones, se ha mantenido en Visual C# 2005, pero no en Visual Studio .NET 2002 y Visual Studio .NET 2003. Si por alguna razón, debe trabajar con alguno de estos entornos, debe saber que esta opción no está disponible para las versiones comentadas.
- Otra característica que debemos conocer de nuestro entorno de desarrollo, es la capacidad de anclar o fijar una ventana de las comentadas anteriormente o de permitir que se haga visible cuando acercamos el puntero del mouse sobre ella. Esta opción es la que puede verse en la figura.



Opción de ocultar o mostrar la ventana seleccionada en Visual Studio 2005

Nótese que al presionar sobre el icono indicado en la figura, haremos que esta ventana quede fija en el entorno de desarrollo. Cuando pulsamos este icono, la ventana queda fija y queda representado por un icono como el que se muestra en la siguiente figura.



Icono para ocultar o mostrar la ventana seleccionada cuando se encuentra en modo anclado

- Algo que oculta el entorno de Visual Studio 2005 por defecto, son las denominadas clases parciales.

Se trata de una nueva característica añadida a .NET 2.0 y por lo tanto a Visual C# 2005, que permite separar o partir una clase en varias porciones de código.

La explicación ruda de esto, es que el programador puede tener dos ficheros de código fuente independientes, que posean el mismo nombre de clase.

Para indicar que pertenece a la misma clase, ésta debe tener la palabra clave `partial` como parte de su definición para indicar que es una clase parcial.

Un ejemplo que aclare esto es el siguiente:

```
partial class Class1
{
    public int Accion1()
    {
        return 1;
    }
}

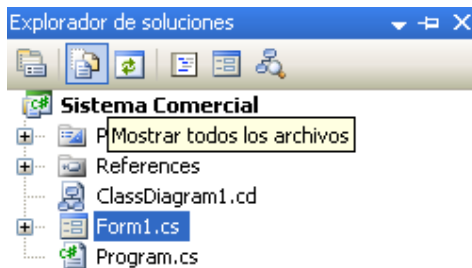
partial class Class1
{
    public int Accion2()
    {
        return 2;
    }
}
```

El comportamiento de la clase es el de una única clase, por lo que su declaración y uso es como el de cualquier clase normal, tal y como se indica en el siguiente código:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click( object sender, EventArgs e )
    {
        Class1 miClase = new Class1();
        MessageBox.Show(miClase.Accion1() + "\n" + miClase.Accion2());
    }
}
```

De todas las maneras, el entorno nos oculta muchas veces las clases parciales de una aplicación.

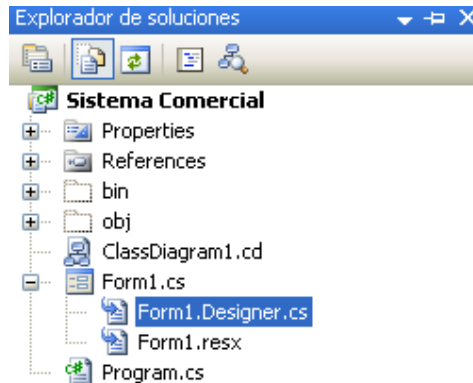
Para ello, seleccionaremos la opción `Mostrar todos los archivos` de la ventana Explorador de soluciones como se indica en la figura.



Icono u opción para mostrar todos los archivos del proyecto

De esta manera, podremos acceder a los archivos y recursos del proyecto, incluidas las clases parciales como se indica en la figura.

En el archivo Form1.Designer.cs estará el código utilizado por el diseñador de formularios de Windows Forms, en el que se incluye la declaración de todos los controles y controladores de eventos que hemos definido en nuestro proyecto.

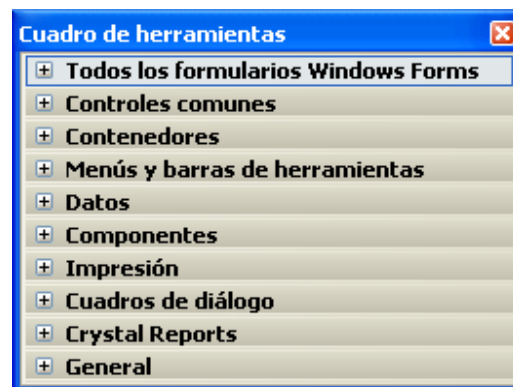


Clase parcial mostrada en los archivos del proyecto

3.7.- Controles Windows Forms

Dentro del entorno de desarrollo de Visual Studio 2005, nos encontramos un enorme conjunto de librerías y controles que podemos utilizar en nuestras aplicaciones Windows. Dependiendo del tipo de aplicación que llevemos a cabo, el entorno habilitará los controles correspondientes para cada tipo de aplicación. En nuestro caso, nos centraremos en los controles más habituales de Windows, e indicaremos como utilizarlos en nuestros desarrollos.

En nuestro entorno de desarrollo, encontraremos diferentes grupos de controles o componentes dispuestos de ser utilizados. En la siguiente figura encontraremos los grupos de controles y componentes más habituales.

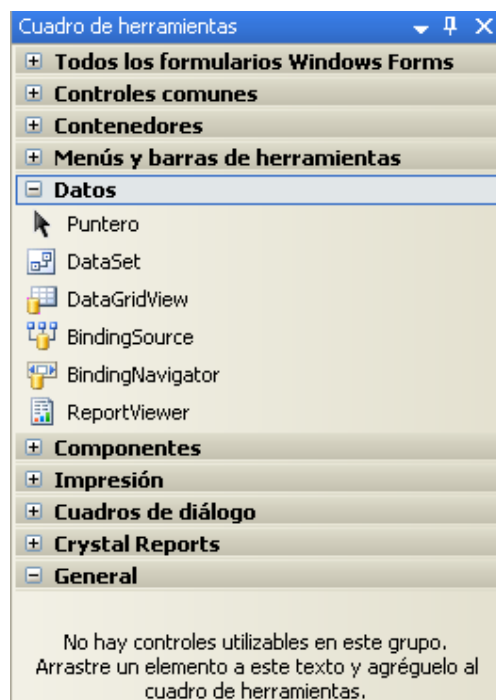


Grupos de controles en Visual Studio 2005

Estos controles se dividen en los grupos representados en la figura anterior. A continuación veremos los más representativos.

Datos

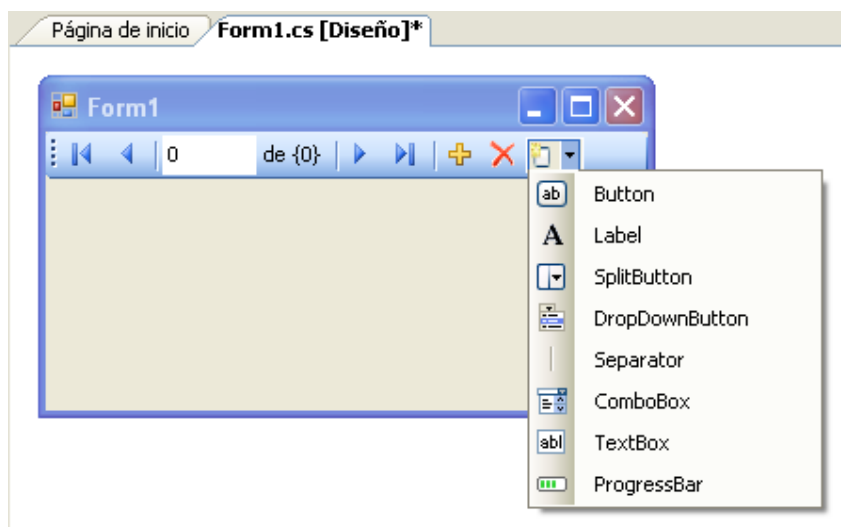
El grupo Datos corresponde con el grupo que tiene relación directa con los componentes de acceso a datos.



Controles Datos en Visual Studio 2005

Para muchos desarrolladores, los controles, componentes y métodos de acceso a datos, contiene dentro de sí un especial misterio, es como el Santo Grial de la programación. Casi siempre nos atascamos ahí, siempre en el mismo sitio. Pero no se preocupe ni lo más mínimo por ello, aprenderemos a utilizarlos a base de práctica, y lo que es más importante, los dominaremos rápidamente. Solo como curiosidad y por ahora, le presentaré uno de los componentes más destacables en Visual Studio 2005, por su semejanza con otro muy utilizado en "otros" entornos de desarrollo, estoy hablando del control y componente BindingNavigator que usaremos frecuentemente en nuestras aplicaciones con acceso a fuentes de datos.

Este control insertado en un formulario Windows.



Control BindingNavigator insertado en un formulario Windows en Visual Studio 2005

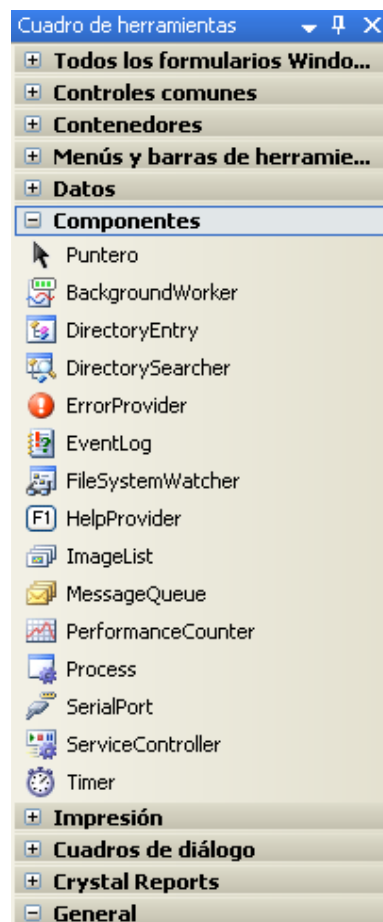
Como puede observar, este control, tiene un aspecto muy similar al del famoso *Recordset* de Visual Basic 6 o al *DataNavigator* de Borland. Lógicamente, este control tiene un aspecto mucho más vistoso y moderno, pero es uno de los controles estrella de Visual Studio 2005, ya que en Visual Studio .NET 2002 y Visual Studio .NET 2003 no existía este control en el entorno.

Visual Studio 2005 sí que nos trae sin embargo, la novedad del control *BindingNavigator*.

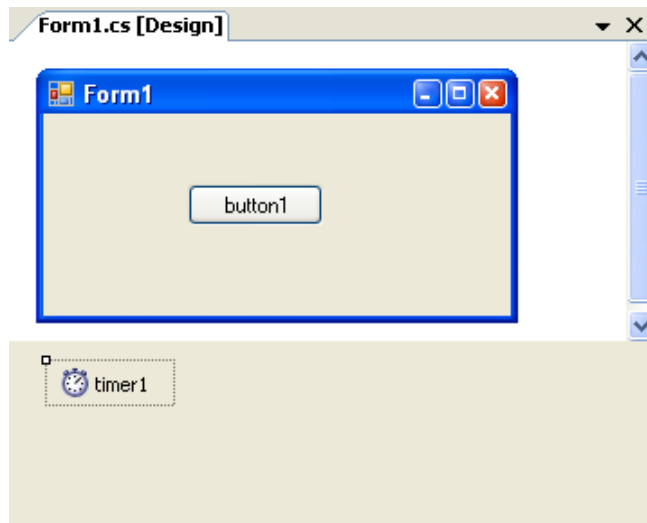
3.8.- Componentes

Windows Forms incluye un conjunto de componentes muy nutrido y variado. Algunos de estos componentes, han sido mejorados y otros ampliados. Los componentes son como controles no visibles, o dicho de otra forma, son controles que realizan ciertas tareas, pero no tienen un interfaz que mostrar, como puede ser el caso de un botón o una caja de textos.

Por ejemplo, el componente *Timer* nos permite recibir una notificación cada x tiempo, pero no muestra nada al usuario de nuestra aplicación. Si hacemos doble clic sobre el componente *Timer* para insertarlo en el formulario, éste quedará dispuesto en la parte inferior del formulario como se indica en la siguiente figura.



Componentes de Windows Forms



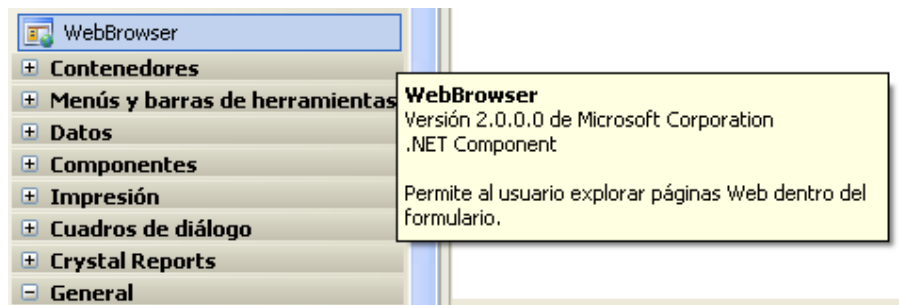
Control Timer insertado en un formulario de Visual C# 2005

Este tipo de componentes no son visibles en tiempo de ejecución.

3.9.- Otros controles a tener en cuenta

Dentro del entorno de Visual Studio 2005 y en .NET en general, se han añadido una serie de controles nuevos que conviene comentar.

Uno de estos controles, se llama *WebBrowser*.



Control WebBrowser en el Cuadro de herramientas

Este control es la representación de un control específico para mostrar contenido XML o contenido HTML, como si de una página Web se tratara.

Sirva el siguiente ejemplo de código fuente para demostrar como usar el control y como se muestra dicho control en una aplicación Windows.

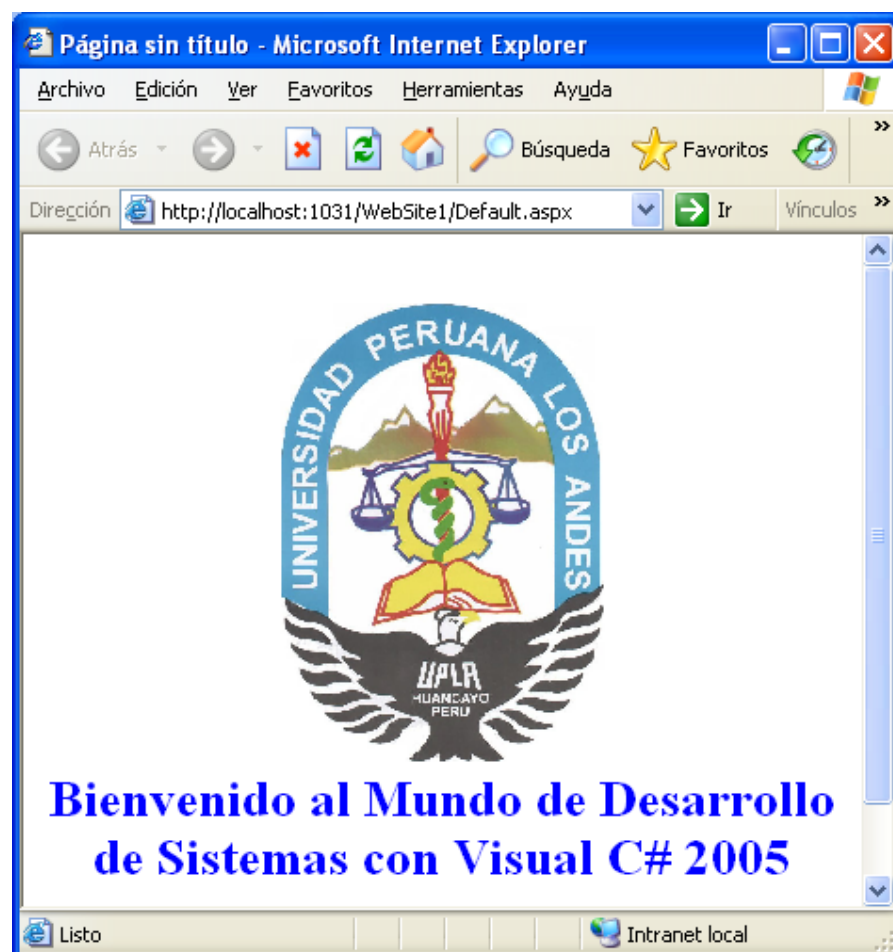
El código de la aplicación quedaría como se detalla a continuación:

```
public partial class Form1 : Form
{
    private void Form1_Load( object sender, EventArgs e )
    {
        this.webBrowser1.Navigate("http://localhost:1031/WebSite1/Default.aspx");
    }
}
```

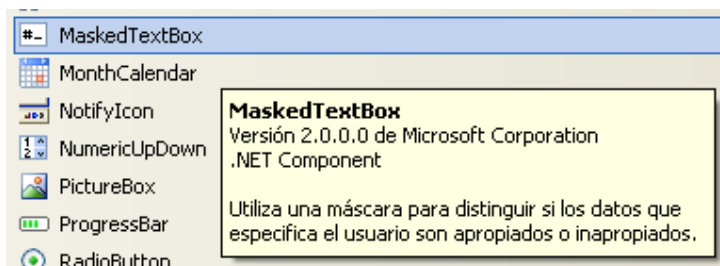
Nuestro ejemplo en ejecución es el que se muestra a continuación.



Control WebBrowser en ejecución

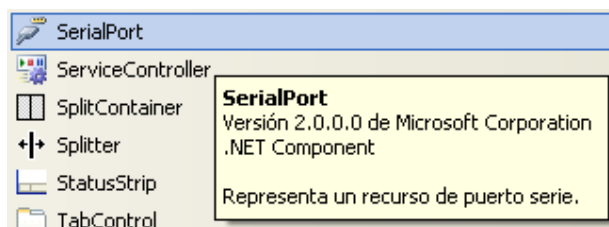


Hay más controles que representan una novedad para el desarrollador de .NET, como puede ser por ejemplo, el control **MaskedTextBox**.



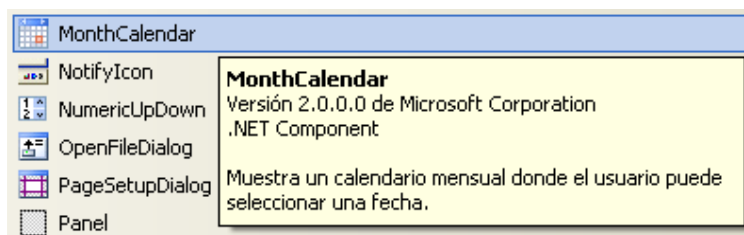
Control MaskedTextBox en Visual C# 2005

Sin embargo, hay otros controles clásicamente demandados por los desarrolladores, como los controles de accesos a puertos COM y puertos serie, como es el caso del control **SerialPort**.



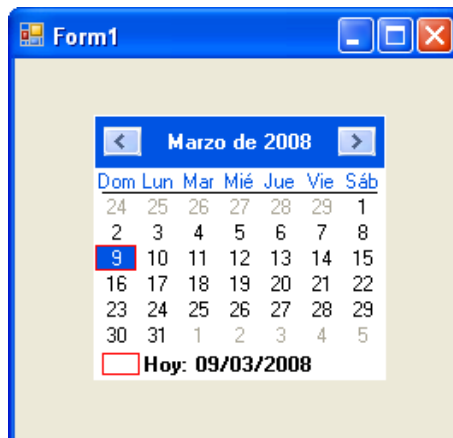
Control SerialPort en Visual C# 2005

No es cuestión de repasar cada uno de los controles que el programador puede encontrar en Visual Studio 2005, sin embargo, no me gustaría dejar de comentar, uno de los controles más usados y útiles para las aplicaciones Windows, que tiene a su vez su equivalente para el desarrollo de aplicaciones Web en ASP.NET. Me refiero al control .



Control MonthCalendar en Visual C# 2005

Este control, que se muestra en la siguiente figura cuando lo insertamos en un formulario, es un control que nos facilita la entrada de fechas en el sistema y permite asegurarnos, que la fecha seleccionada es una fecha válida.



Control MonthCalendar insertado en un formulario Windows



RESUMEN

Las aplicaciones Windows del entorno de trabajo .NET son aplicaciones basadas en uno o mas formularios Windows. Estas aplicaciones se dirigen a computadoras que pueden ejecutar código asociado a un formulario y proporcionan un entorno al usuario. Para el desarrollo de los mismos Visual Studio proporciona un conjunto de herramientas integradas de alto grado de funcionalidad y de uso nada complicado. Estas herramientas proporcionan métodos acelerados de desarrollo de aplicaciones basadas en Windows Forms.



BIBLIOGRAFÍA RECOMENDADA

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.



NEXO

En la unidad TEMÁTICA realizamos una conexión desde un proyecto Visual Studio C# a una base de datos de SQL Server.

Demostramos la utilidad del control WebBrowser para mostrar el contenido de página Web de nuestro servidor local. El mismo que puede aplicarse a cualquier dirección de Internet.



ACTIVIDAD

Debo asumir que para el desarrollo de sistemas de información usted debe tener conocimientos de modelado y diseño de bases de datos. En la organización que está desarrollando el sistema de información debe Modelar, Diseñar una base de datos e Implementarla en Microsoft SQL Server.



AUTOEVALUACIÓN FORMATIVA

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

- 1.- Diseñar una aplicación basada en Windows que muestre la fecha y hora del sistema. Debe actualizarse en forma automática.
- 2.- Diseñar una aplicación que utilice el control MonthCalendar para seleccionar su fecha de nacimiento y le muestre un mensaje determinando su edad en años, meses y días.
- 3.- Diseñar una aplicación Windows que permita ingresar un importe económico ingresado en un tipo de moneda X y lo convierta a otro tipo de moneda Y. Utilice ComboBox.
- 4.- Diseñar una aplicación Windows que simule un navegador de Internet.

Unidad Temática IV

CREACIÓN DE CONTROLES

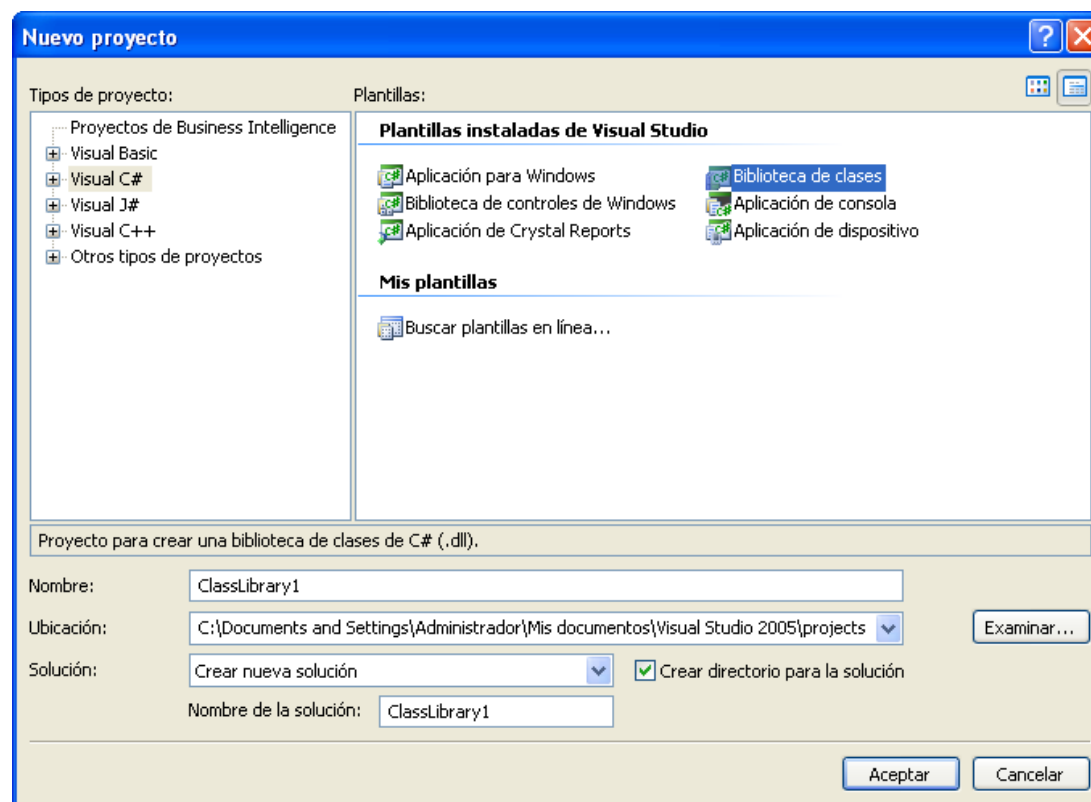
4.1.- Clases y controles personalizados

Ya hemos visto la diferencia más genérica entre un componente y un control, pero aún no sabemos como desarrollar nuestros propios controles en Visual Studio 2005.

En primer lugar y antes de adentrarnos en la creación de nuestros propios controles con Visual Studio 2005, debo indicarle que debemos obviar todo lo relacionado con los ActiveX OCX y ActiveX en general.

En Visual Studio 2005, la palabra ActiveX ya no existe. El modelo de programación ha cambiado y por eso, los componentes y controles se generan ahora siguiendo otras normas que aprenderemos a utilizar de forma inmediata.

Iniciaremos Visual Studio 2005 y seleccionaremos un proyecto de tipo Biblioteca de clases. En el nombre de proyecto, podemos indicarle el nombre que deseemos tal y como se muestra en la figura, y a continuación presionaremos el botón OK.



Selección de nuevo proyecto Biblioteca de clases en Visual C# 2005

La diferencia mayor que reside entre el desarrollo de componentes y controles en .NET, es que en lugar de heredar de la clase Component como en el caso de la creación de los componentes, se ha de heredar de la clase Control o System.Windows.Forms.UserControl.

El tipo de proyecto seleccionado no posee por defecto como ocurre con los controles ActiveX, de la superficie contenedora sobre la cuál podremos insertar otros controles o realizar las acciones que consideremos pertinentes para crear así nuestro control personalizado. En este caso, la superficie contenedora la deberemos crear añadiendo las referencias necesarias a nuestro programa.

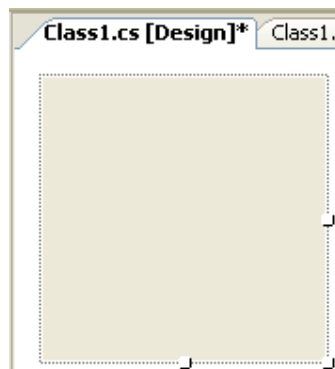
Haga clic con el botón secundario del mouse sobre el proyecto o solución de la ventana Explorador de soluciones y a continuación, seleccione la opción Propiedades del menú emergente.

A continuación, agregue las referencias a las librerías de clases System.Drawing y System.Windows.Forms.

Por último, escriba las siguientes instrucciones básicas.

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing;
namespace ClassLibrary1
{
    public class Class1 : UserControl
    {
    }
}
```

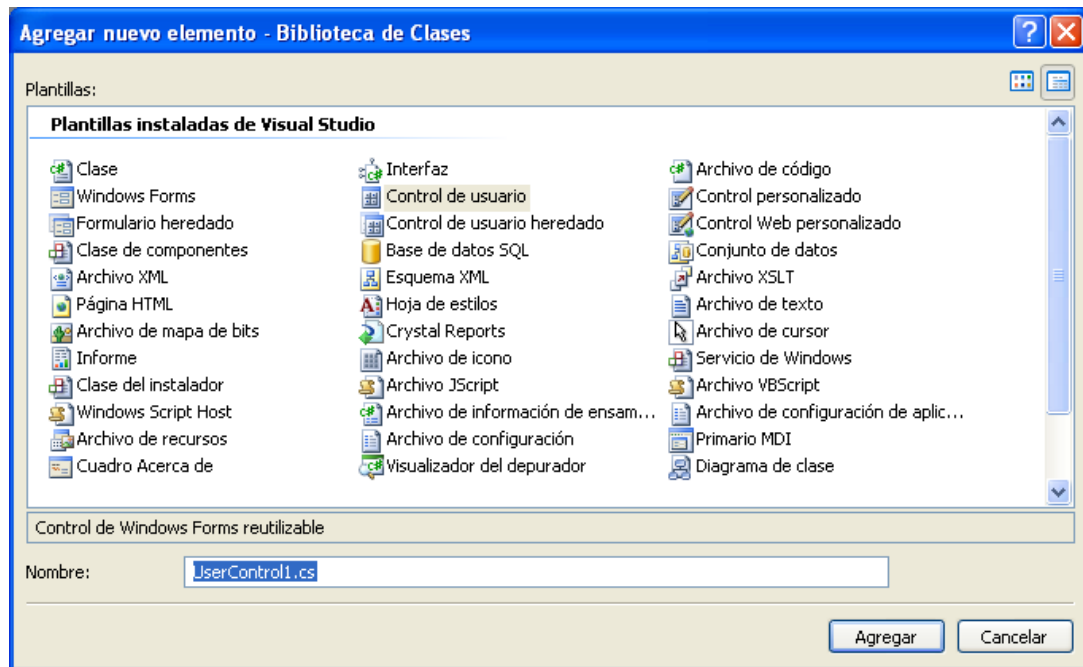
En este punto, nuestra clase habrá sido transformada en la clase contenedora de un control que es la que puede verse en la figura.



Superficie contenedora por defecto de un control

Aún así, sino quiere realizar esta acción, otra forma de tener lista la superficie contenedora del control, es la de eliminar la clase del proyecto y presionar el botón secundario del mouse sobre la ventana del Explorador de soluciones y seleccionar la opción de **Agregar > Nuevo elemento...** del menú emergente.

De las opciones que salen, deberíamos seleccionar entonces la plantilla Control de usuario.



Seleccionar un control de usuario

Sin más dilación, añadiremos sobre la superficie del control contenedor, (al que habremos cambiado el nombre a MiControl), dos controles Label y dos controles TextBox.

Debido a que al añadirle los controles tendremos mucho código que simplemente sirve para el diseño del interfaz de usuario, podemos hacer lo que el propio diseñador hace: dividir la clase en dos partes o clases parciales, de esta forma conseguiremos el propósito de las clases parciales, (al menos en cuanto al diseño de formularios y controles de usuario se refiere), que es, como ya comentamos anteriormente, separar el código de diseño del código que nosotros vamos a escribir.

Para ello, añadiremos un nuevo elemento del tipo Class, al que le daremos el nombre MiControl.Designer.cs, y en esa clase pegaremos el código que actualmente hay en la clase, pero añadiendo la partícula parcial antes de class, con idea de que el compilador de Visual C# 2005 sepa que nuestra intención es crear una clase parcial.

En el nuevo archivo tendremos el siguiente código:

```
namespace Biblioteca_de_Clases
{
    partial class MiControl
    {
        /// <summary>
        /// Variable del diseñador requerida.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén utilizando.
        /// </summary>
        /// <param name="disposing"> true si los recursos administrados se deben eliminar; false en
        /// caso contrario, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
            }
        }
    }
}
```

```

        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Código generado por el Diseñador de componentes

/// <summary>
/// Método necesario para admitir el Diseñador. No se puede modificar
/// el contenido del método con el editor de código.
/// </summary>
private void InitializeComponent()
{
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
    this.label1.Location = new System.Drawing.Point(11, 15);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(62, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = "Usuario : ";
    //
    // label2
    //
    this.label2.AutoSize = true;
    this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
    this.label2.Location = new System.Drawing.Point(11, 41);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(83, 13);
    this.label2.TabIndex = 1;
    this.label2.Text = "Contraseña : ";
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(94, 12);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(172, 20);
    this.textBox1.TabIndex = 2;
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(94, 38);
    this.textBox2.Name = "textBox2";
    this.textBox2.PasswordChar = '*';
    this.textBox2.Size = new System.Drawing.Size(172, 20);
    this.textBox2.TabIndex = 3;
    //
    // MiControl
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.textBox1);
    this.Controls.Add(this.label2);

```

```

        this.Controls.Add(this.label1);
        this.Name = "MiControl";
        this.Size = new System.Drawing.Size(284, 75);
        this.Load += new System.EventHandler(this.MiControl_Load);
        this.ResumeLayout(false);
        this.PerformLayout();
    }

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
}
}

```

Realmente es casi lo mismo que tenemos actualmente, tan solo tendremos que añadir la instrucción parcial, (no hace falta que esté declarada como public, ya que en el otro "trozo" tendremos declarada esta clase como pública), además del constructor de la clase, desde el que llamaremos al método que se encarga de toda la inicialización: **InitializeComponent()**.

A continuación, escribiremos el siguiente código, en el archivo que teníamos originalmente (MiControl.cs):

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

namespace Biblioteca_de_Clases
{
    public partial class MiControl : UserControl
    {
        private bool _Acceso;

        [Category("Acceso"),
        Description("Indica si se permite o no el acceso"),
        DefaultValue(false),
        ReadOnly(true)]

        public MiControl()
        {
            InitializeComponent();
        }

        public bool Acceso
        {
            get { return _Acceso; }
            set { _Acceso = value; }
        }

        public void Validar()
        {
            if (textBox1.Text == "ejemplo" & this.textBox2.Text == "ejemplo")
                _Acceso = true;
            else
                _Acceso = false;
        }
    }
}

```



```

    }

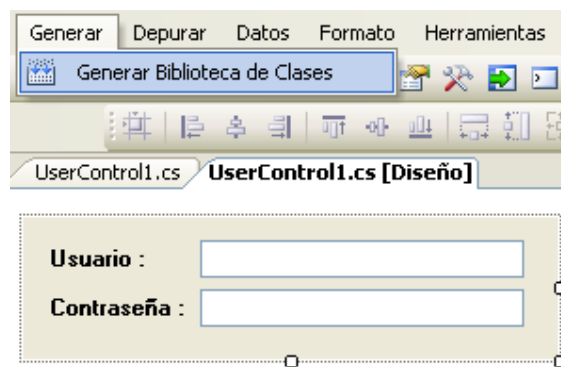
    private void MiControl_Load(object sender, EventArgs e)
    {
        _Acceso = false;
    }
}

```

Observando el código, vemos que hemos creado una propiedad **Acceso** que nos permitirá saber si un usuario y contraseña han sido validadas o no.

Ahora nuestro control, está listo ya para ser compilado y probado en una aplicación Windows.

Para compilar nuestro control, haga clic en el menú Generar > Generar Solución como se muestra en la figura.



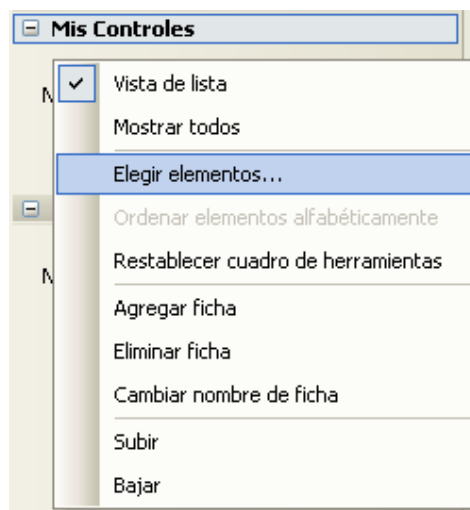
Opción para compilar nuestro control

En un proyecto que esté desarrollando una aplicación cualquiera, podrá utilizar el control que acabó de implementar.

Acuda a la barra de herramientas y busque el control que hemos creado y compilado para insertarlo en el formulario Windows.

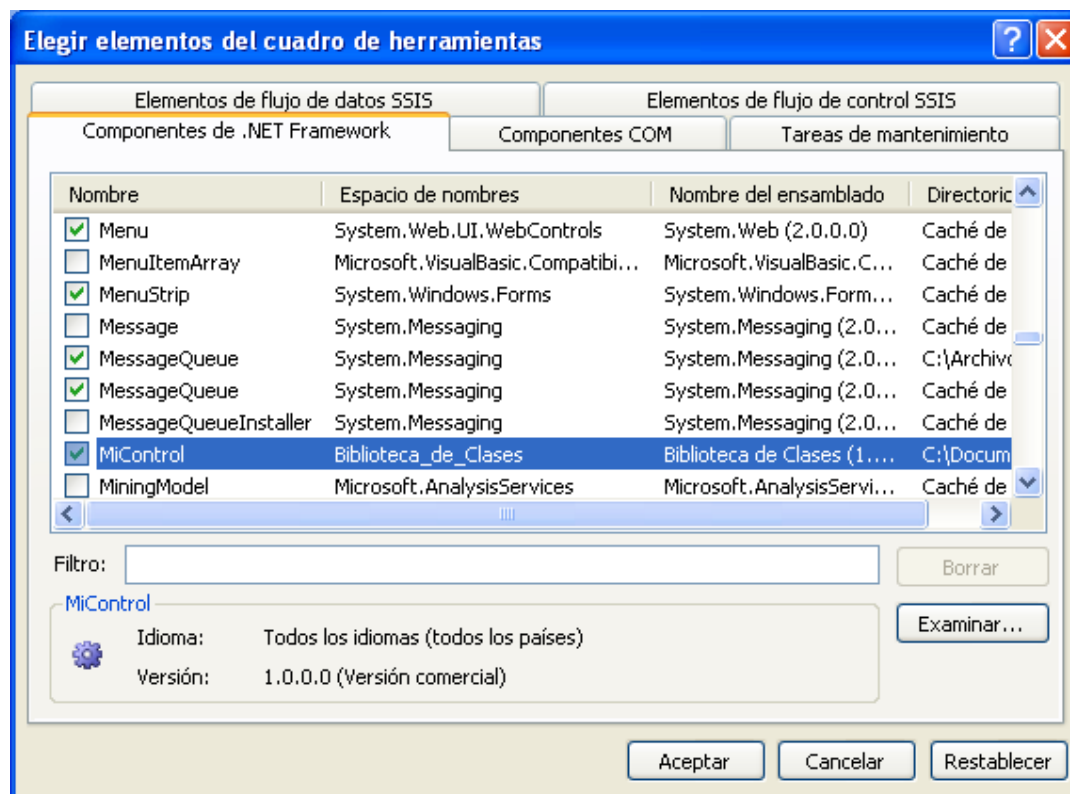
Sino aparece en la barra de herramientas, (que debería aparecer en el grupo **ClassLibrary1**), deberá añadirlo de la siguiente manera.

Haga clic sobre la barra de herramientas y seleccione la opción **Elegir Elementos...** del menú emergente que aparece en la figura.



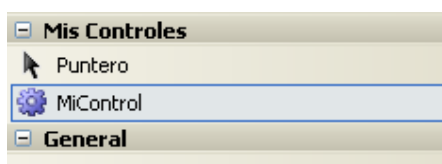
Opción para añadir un control o componente a la barra de herramientas

Aparecerá una ventana para buscar el control ya compilado en el disco duro. Presionaremos el botón **Examinar...** y buscaremos nuestro control para seleccionarlo. Una vez hecho esto, tal y como se indica en la figura, haremos clic sobre el botón **OK**.



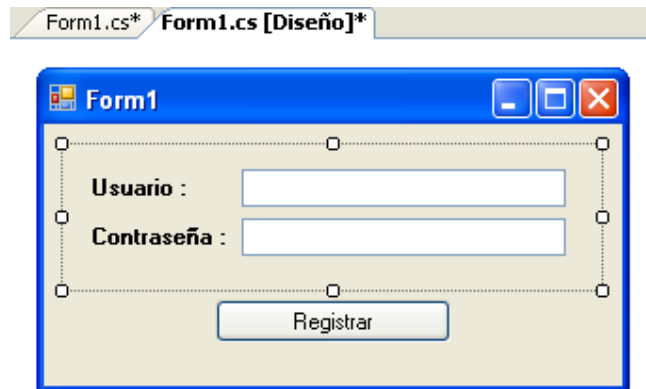
Selección del control compilado anteriormente

Nuestro control quedará insertado en la barra de herramientas como se muestra en la figura.



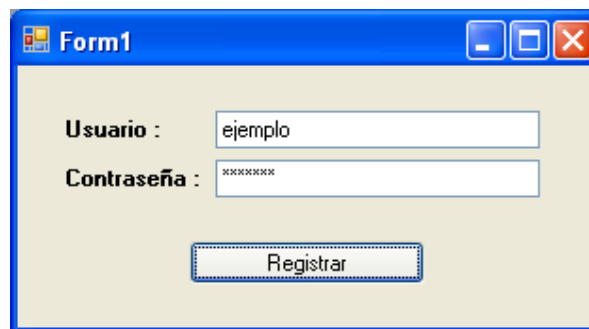
Control insertado en la barra de herramientas

Para insertar el control en el formulario, haremos doble clic sobre él. Este quedará dispuesto en el formulario Windows como se indica en la figura.



Control insertado en el formulario Windows de prueba

Nuestro formulario Windows de prueba en ejecución con el control insertado en él, es el que puede verse en la figura.



Formulario Windows de prueba en ejecución con el control insertado

Como ha podido comprobar, la creación de controles en Visual C# 2005, tampoco requiere de una gran habilidad o destreza, y su similitud con la creación de componentes es enorme. De hecho, todo se reduce en el uso y programación de una clase con la salvedad de que dentro de esa clase, indicamos si se trata de una clase como tal, la clase de un componente o la clase de un control.

4.2.- Controles contenedores

Cuando trabajamos con controles, surgen muchas veces muchas dudas de carácter habitual. Definamos que hay dos tipos de controles, los controles contenedores y los controles no contenedores.

La diferencia entre ambos es que los controles contenedores, pueden como su propia palabra dice, contener otros controles dentro de éste. Los controles no contenedores no pueden.

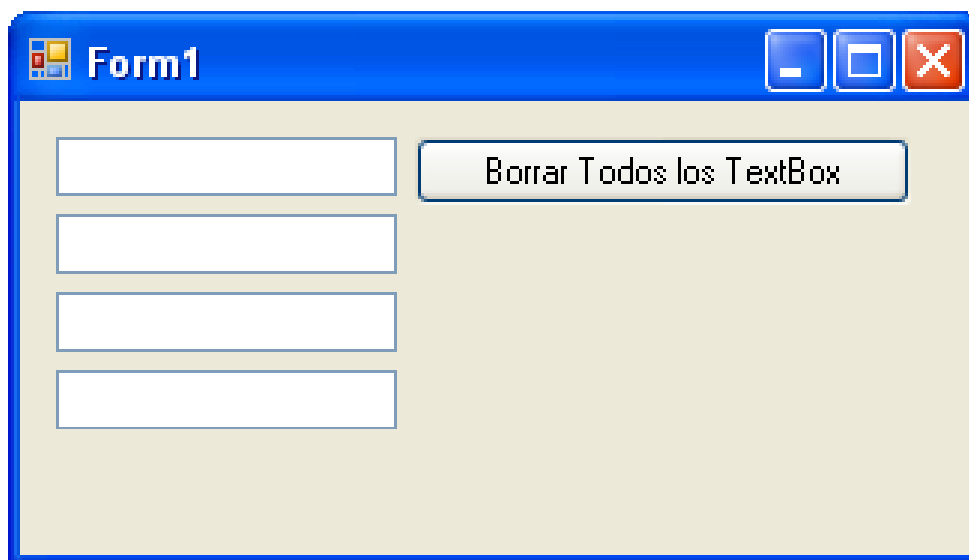
Un claro ejemplo de control contenedor, es el control `TabControl`, el cuál permite incluir dentro de él otros controles, incluso controles contenedores a su vez.

Un ejemplo de control no contenedor es el control `Button` por ejemplo. Si intenta poner un control `TextBox` encima de un control `Button`, observará que no puede.

Cuando un control se inserta dentro de un contenedor (no olvidemos que el formulario `Windows` es otro contenedor), éste control queda alojado dentro de su contenedor, de tal manera que si movemos el contenedor, estaremos moviendo también el contenido de éste.

El problema sin embargo cuando trabajamos con controles, viene cuando deseamos recorrer los controles de un formulario. Esto es muy habitual cuando deseamos borrar el contenido de todos los controles `TextBox` de un formulario `Windows` por ejemplo.

Dentro de un formulario y supuesto que tengamos 4 controles TextBox insertados dentro de él, y luego un botón que nos permita eliminar el contenido de cada caja de texto, tal y como se indica en la figura, deberíamos escribir un código similar al que veremos a continuación:



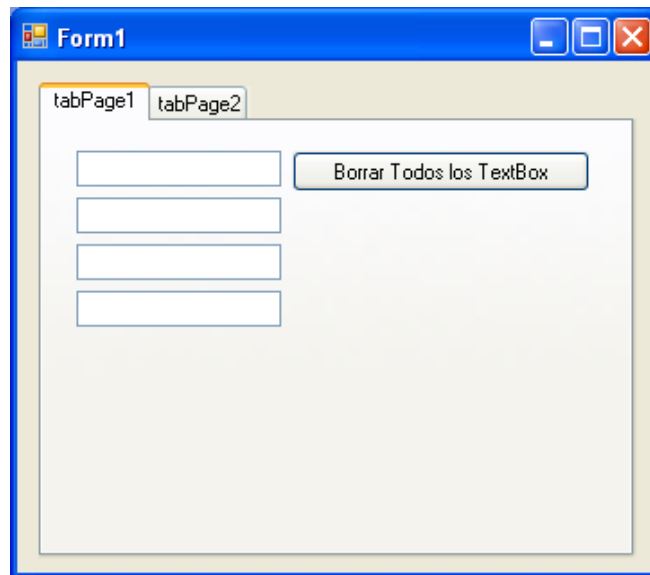
Formulario Windows de ejemplo con sus controles insertados

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click( object sender, EventArgs e )
    {
        foreach( object obj in Controls )
        {
            if( obj is TextBox )
            {
                ((TextBox)obj).Text = "";
            }
        }
    }
}
```

Observando este código, vemos que lo que hacemos no es otra cosa que recorrer los objetos como controles dentro del formulario y miramos si se trata de un control de tipo TextBox. En este caso, modificamos la propiedad Text para dejarla en blanco. Ejecute este ejemplo con la tecla de función F5, escriba algo en las cajas de texto y pulse el botón. Observará que logramos conseguir nuestro objetivo de borrar el contenido de todos los controles TextBox.

A continuación, lo que haremos será añadir un control TabControl a nuestro formulario Windows, y dentro de él, añadiremos los cuatro controles TextBox y el control Button anteriormente comentados, tal y como se muestra en la siguiente figura.



Controles insertados dentro de un control contenedor como el control TabControl

El código anterior no hará falta modificarlo, por lo que ejecutaremos la aplicación nuevamente presionando el botón F5, escribiremos algo de texto en las cajas de texto y pulsaremos el botón como hicimos antes. Observaremos que en este caso, los controles TextBox no se han quedado en blanco como antes.

El contenedor no es el formulario Windows, sino el control TabControl. Dicho control es tratado en el bucle anterior del control como control dependiente del formulario Windows, pero los controles TextBox y el propio control Button, quedan encerrados dentro de un ámbito contenedor diferente.

Para solventar este problema, deberemos recorrer los controles del contenedor. Una forma de solventar esto es de la siguiente manera:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click( object sender, EventArgs e )
    {
        for( byte i = 0; i < tabControl1.TabPages.Count; i++ )
            foreach( object obj in tabControl1.TabPages[i].Controls )
                if( obj is TextBox )
                    ((TextBox)obj).Text = "";
    }
}
```

De esta manera, recorreremos todos los controles que residen en las páginas de un control TabControl y si se trata de un control TextBox, modificaremos la propiedad Text correspondiente.



Por esta razón, cuando trabajamos con controles dentro de un formulario y queremos actuar sobre un conjunto de controles determinado, debemos tener en cuenta entre otras cosas, si se trata de un conjunto de controles o un control simplemente, se encuentra dentro de un control contenedor o fuera de él.

Sin embargo, para resolver el problema de recorrer todos los controles de un formulario, estén o no dentro de un control contenedor, lo mejor es ampliar la función anterior y hacerla recursiva, de modo que permita recorrer todos los controles del formulario, estén o no dentro de un contenedor.

En el mismo proyecto, podemos añadir nuevos TextBox a la otra ficha del TabControl, además uno en el propio formulario, (para que no esté dentro del TabControl).

Rellene algo de contenido en los controles TextBox y ejecute el código:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click( object sender, EventArgs e )
    {
        VaciarTextBox(this);
    }
    private void VaciarTextBox( Control Parent )
    {
        foreach( Control obj in Parent.Controls )
        {
            if( obj.Controls.Count > 0 )
                VaciarTextBox(obj);
            if( obj is TextBox )
                ((TextBox)obj).Text = "";
        }
    }
}
```

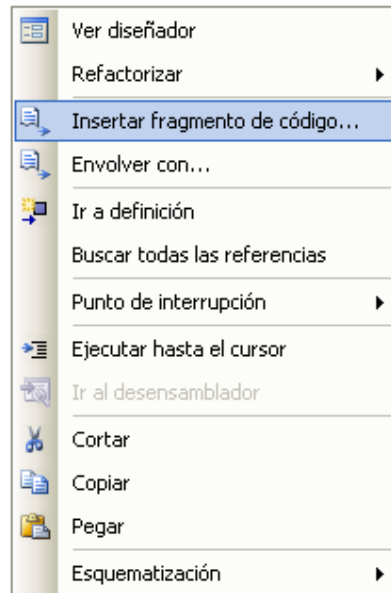
4.3.- Generación de código rápido

Otra consideración a tener en cuenta a la hora de trabajar con código y a la hora por lo tanto, de desarrollar nuestros propios controles, componentes, clases, etc., es la ayuda que nos proporciona el entorno cuando deseamos escribir determinadas funciones rápidas de código que son bastante habituales.

Posiciónese sobre el código de Visual C# 2005 y haga clic con el botón secundario del mouse y seleccione la opción Insertar fragmento de código... como se indica en la siguiente figura.

```
private void button1_Click(object sender, EventArgs e)
{
}

```



Opción de recortes de código de Visual Studio 2005

```
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            switch (switch_on)
            {
                default:
            }
        }
    }
}

```

El código generado por el snippet



RESUMEN

Los formularios Windows existen dentro de las aplicaciones Windows. Se puede crear esqueletos de una aplicación desde una plantilla o controles de usuario personalizados dentro de Visual Studio. Las aplicaciones Windows se utilizan en entornos donde puede resultar adecuado sacar partido de la capacidad de proceso de las estaciones cliente. Cuando se desarrollan sistemas basados en plantillas de aplicaciones Windows, a menudo se utilizan formularios Windows para recoger las entradas de los usuarios y mostrar información a los usuarios.



BIBLIOGRAFÍA RECOMENDADA

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.



NEXO

En la unidad TEMÁTICA muestra como el desarrollador de sistemas puede implementar sus plantillas – controles de usuario personalizados.



ACTIVIDAD

En la organización que está desarrollando el sistema de información implemente plantillas que utilizará en todo su sistema. Por ejemplo plantilla para formularios, cuadros de Texto, Etiquetas, etc.



AUTOEVALUACIÓN FORMATIVA

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

- 1.- Diseñar una aplicación basada en Windows que permita autenticar a un usuario del sistema de información. El usuario debe proporcionar su nombre de usuario y su contraseña. Dado que a la fecha no desarrollamos el tema de interacción con bases de datos. La aplicación debe validar a los usuarios que hayan sido ingresados con anterioridad a una estructura con los elementos: Usuario, Apellidos, Nombres y Contraseña. Debe crear un array de veinte usuarios con la estructura definida.
- 2.- Diseñar una aplicación Windows que permita administrar una tabla de empleados (insertar, Modificar y eliminar datos), la información será almacenada en una estructura que considere los campos Código de empleado, apellido paterno, apellido materno, nombres, dirección, etc.

Unidad Temática V

APLICACIONES CON ACCESO A DATOS

5.1.- Acceso a Datos

En este módulo, aprenderemos a trabajar con datos y fuentes de datos en Visual Studio 2005. ADO.NET es la tecnología principal para conectarse a una base de datos, nos ofrece un alto nivel de abstracción, ocultando los detalles de bajo nivel de la implementación de la base de datos de un fabricante.

Encontrará las cosas más importantes que debe saber, para trabajar con fuentes de datos con Visual Studio 2005.

De esta manera, aprenderá en poco tiempo, a encontrarse cómodo en el entorno de clases de ADO.NET y podrá así, sacar el máximo provecho a sus desarrollos.

A continuación veremos todo lo que debe saber sobre ADO.NET para crear aplicaciones que accedan a fuentes de datos desde Visual Studio 2005.

Comprobará que ahora, es incluso mucho más fácil y rápido, si bien, es necesario conocer el modelo con el que se trabaja para poder saber lo que deberemos hacer en un momento dado.

5.2.- ADO.NET

ADO.NET ha sufrido a lo largo de los últimos años diferentes mejoras y actualizaciones, desde que .NET apareció.

El resumen de las diferentes versiones de ADO.NET podría quedar de la siguiente forma. ADO.NET 1.0 apareció con Microsoft .NET Framework 1.0. Posteriormente, ADO.NET 1.1 sufrió una pequeñas y casi inapreciables actualizaciones con la aparición de Microsoft .NET Framework 1.1. En el caso del entorno Visual Studio 2005, éste trabaja con Microsoft .NET Framework 2.0 y por lo tanto, utiliza ADO.NET 2.0, el cuál añade algunas características nuevas adicionales.

En nuestro caso, nos centraremos única y exclusivamente en ADO.NET como modelo de objetos de acceso a datos para la plataforma .NET de Microsoft, ya que es el mismo para cualquier tipo de versión de ADO.NET.

¿Qué es ADO.NET?

ADO.NET es la tecnología principal para conectarse a un gestor de bases de datos, con un alto nivel de abstracción, lo que nos permite olvidarnos de los detalles de bajo nivel de las bases de datos. Además ADO.NET es una tecnología interoperativa. Aparte del almacenamiento y recuperación de datos, ADO.NET introduce la posibilidad de integrarse con el estándar XML, los datos pueden 'Serializarse' directamente a y desde XML lo que favorece el intercambio de información.

ADO.NET proporciona diferentes clases del nombre de espacio System.Data dentro de las cuáles, destacaremos por encima de todas, la clase DataView, la clase DataSet y la clase DataTable.

Este conjunto de clases de carácter armónico, funcionan de igual forma con la capa inferior que es la que corresponde a los proveedores de acceso a datos con los que podemos trabajar.

Esto facilita el trabajo en n-capas y la posible migración de aplicaciones que utilicen una determinada fuente de datos y deseemos en un momento dado, hacer uso de otra fuente de datos.

¿Qué capas o qué partes hay dentro de ADO.NET?

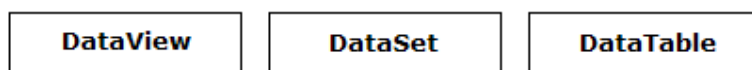
Dentro de ADO.NET tenemos dos partes importantes.

La primera de ellas es la que corresponde con el nombre de espacio System.Data y que constituye los objetos y clases globales de ADO.NET.

La otra parte es la que corresponde con los objetos que permiten el acceso a datos a una determinada fuente de datos desde ADO.NET y que utilizan así mismo, las clases del nombre de espacio System.Data.

Esta última parte, queda constituida por las clases y objetos de los diferentes proveedores de acceso a datos como se muestra en la figura.

System.Data



Proveedor de Acceso a Datos



Visión general de las clases de ADO.NET

Para resumir de alguna forma lo que estamos comentando, diremos que el trabajo de conexión con la base de datos, la ejecución de una instrucción SQL determinada, una vista, etc., la realiza el proveedor de acceso a datos.

Recuperar esos datos para tratarlos, manipularlos o volcarlos a un determinado control o dispositivo, es acción de la capa superior que corresponde con el nombre de espacio System.Data.

A continuación veremos todo esto con más detalle y comprenderemos de una forma más clara cada una de las partes que componen el modelo de trabajo con ADO.NET.

¿Qué nos permite realmente ADO.NET cuando trabajamos con XML?

El entorno de Microsoft .NET Framework nos proporciona el trabajo con estándares y con ello, la posibilidad de trabajar con diferentes tipos de aplicaciones, entornos, sistemas operativos y lenguajes sin necesidad de conocer lo que hay al otro lado de nuestra aplicación. XML es sin lugar a dudas, el lenguaje de etiquetas por excelencia, válido para llevar a cabo esta tarea sin tener un impacto relevante cuando trabajamos con diferentes soluciones en entornos dispares.

Tanto la posibilidad de trabajar con Servicios Web XML como con documentos e información en XML, sobre todo al trabajar con fuentes de datos en ADO.NET, nos proporciona a los desarrolladores las posibilidades necesarias que nos permite hacer que la información con la que trabajamos, pueda ser tratada entre diferentes sistemas o entornos, sin que por ello nos preocupemos de lo que hay al otro lado.



5.3.- System.Data

Las clases del nombre de espacio System.Data son bastantes extensas y variadas.

Quizás las clases más importantes son la clase DataView, la clase DataSet y la clase DataTable.

La clase DataSet

El DataSet es una representación de datos residente en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene. El DataSet contiene en sí, un conjunto de datos que han sido volcados desde el proveedor de datos.

Debemos tener en cuenta que cuando trabajamos con DataSets, el origen de datos no es lo más importante, ya que éste, puede ser cualquier tipo de origen de datos. No tiene porqué ser una base de datos.

Un DataSet contiene colecciones de DataTables y DataRelations.

El DataTable contiene una tabla o tablas, mientras que la DataRelation contiene las relaciones entre las DataTables.

Sin embargo, no es necesario especificar todo esto hasta el último detalle como veremos más adelante.

La clase DataView

Este objeto nos permite crear múltiples vistas de nuestros datos, además de permitirnos presentar los datos.

Es la clase que nos permite representar los datos de la clase DataTable, permitiéndonos editar, ordenar y filtrar, buscar y navegar por un conjunto de datos determinado.

La clase DataTable

Este objeto nos permite representar una determinada tabla en memoria, de modo que podamos interactuar con ella.

A la hora de trabajar con este objeto, debemos tener en cuenta el nombre con el cuál definamos una determinada tabla, ya que los objetos declarados en el DataTable es sensitivo a mayúsculas y minúsculas.

Un ejemplo práctico

El siguiente ejemplo práctico, nos enseña a utilizar un DataSet y nos muestra como podemos acceder a los objetos que dependen de un DataSet para recuperar por ejemplo, los campos y propiedades de una determinada tabla o tablas.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
```

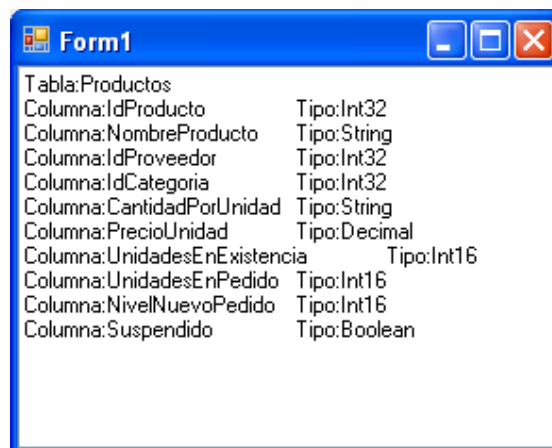
```

{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    string Conexion = "Data Source=SERVIDOR;Initial Catalog=Sistema_Comercial;Integrated
Security=True";
    DataSet MiDataSet = new DataSet();
    SqlDataAdapter MiAdaptador = new SqlDataAdapter("SELECT * FROM Productos",
Conexion);
    MiAdaptador.Fill(MiDataSet);
    foreach (DataTable tabla in MiDataSet.Tables)
    {
        textBox1.Text += "Tabla:" + tabla.TableName + "\n";
        foreach (DataColumn columna in tabla.Columns)
        {
            textBox1.Text += "Columna:" + columna.ColumnName + "\t" + "Tipo:" +
columna.DataType.Name + "\n";
        }
    }
}
}

```

Nuestro ejemplo en ejecución es el que se muestra en la figura.



Ejemplo en ejecución del uso de DataSet, DataTable y DataColumn

5.4.- Los proveedores de acceso a datos

Los proveedores de acceso a datos es la capa inferior de la parte correspondiente al acceso de datos y es la responsable de establecer la comunicación con las fuentes de datos.

En este conjunto de nombres de espacio, encontraremos casi siempre las clases Connection, Command, DataAdapter y DataReader como las clases más generales, las cuales nos permiten establecer la conexión con la fuente de datos.

Proveedores de acceso a datos de .NET Framework

Dentro del entorno .NET Framework, encontramos un nutrido conjunto de proveedores de acceso a datos.

Estos son los siguientes:

- ODBC .NET Data Provider

- OLE DB .NET Data Provider
- Oracle Client .NET Data Provider
- SQL Server .NET Data Provider

Estos proveedores de acceso a datos incluidos en Microsoft .NET Framework, los podemos encontrar en los nombres de espacio:

- System.Data.Odbc
- System.Data.OleDb
- System.Data.OracleClient
- System.Data.SqlClient

El proveedor ODBC .NET permite conectar nuestras aplicaciones a fuentes de datos a través de ODBC.

El proveedor OLE DB .NET permite conectar nuestras aplicaciones a fuentes de datos a través de OLE DB.

El proveedor Oracle Client .NET es un proveedor de acceso a datos especialmente diseñado para bases de datos Oracle.

Por último, el proveedor SQL Server .NET es un proveedor de acceso a datos nativo, que nos permite conectar nuestras aplicaciones a fuentes de datos Microsoft SQL Server 7 o posterior. Se trata de un proveedor específico para bases de datos Microsoft SQL Server 7.0, Microsoft SQL Server 2000 y Microsoft SQL Server 2005.

Los proveedores de acceso a datos que distribuye Microsoft en ADO.NET y algunos desarrollados por otras empresas o terceros, contienen los mismos objetos, aunque los nombres de éstos, sus propiedades y métodos, pueden ser diferentes.

Más adelante veremos algún ejemplo, y observará en la práctica cuáles son estas diferencias más destacables.

Otros proveedores de acceso a datos

Si bien el proveedor de acceso a datos es el mecanismo a través del cuál podemos establecer una comunicación nativa con una determinada fuente de datos, y dado que Microsoft proporciona los proveedores de acceso a datos más corrientes, es cierto que no los proporciona todos, si bien, con OLE DB y ODBC, podemos acceder a la inmensa totalidad de ellos.

Sin embargo, hay muchos motores de bases de datos de igual importancia como Oracle, MySQL, AS/400, etc.

En estos casos, si queremos utilizar un proveedor de acceso a datos nativo, deberemos acudir al fabricante o a empresas o iniciativas particulares para que nos proporcionen el conjunto de clases necesarias que nos permitan abordar esta acción.

El objeto Connection

Este objeto es el encargado de establecer una conexión física con una base de datos determinada.

Para establecer la conexión con una determinada fuente de datos, no sólo debemos establecer la cadena de conexión correctamente, sino que además deberemos usar los parámetros de conexión y el proveedor de acceso a datos adecuado.

Con este objeto, podremos además abrir y cerrar una conexión.

El objeto Command

Este objeto es el que representa una determinada sentencia SQL o un Stored Procedure. Aunque no es obligatorio su uso, en caso de necesitarlo, lo utilizaremos conjuntamente con el objeto DataAdapter que es el encargado de ejecutar la instrucción indicada.

El objeto DataAdapter

Este objeto es quizás el objeto más complejo y a la vez complicado de todos los que forman parte de un proveedor de acceso a datos en .NET.

Cuando deseamos establecer una comunicación entre una fuente de datos y un DataSet, utilizamos como intermediario a un objeto DataAdapter.

A su vez, un DataAdapter contiene 4 objetos que debemos conocer:

- SelectCommand es el objeto encargado de realizar los trabajos de selección de datos con una fuente de datos dada.
- En sí, es el que se encarga de devolver y rellenar los datos de una fuente de datos a un DataSet.
- DeleteCommand es el objeto encargado de realizar las acciones de borrado de datos.
- InsertCommand es el objeto encargado de realizar las acciones de inserción de datos.
- UpdateCommand es el objeto encargado de realizar las acciones de actualización de datos.

Los objetos DeleteCommand, InsertCommand y UpdateCommand son los objetos que se utilizan para manipular y transmitir datos de una fuente de datos determinada, al contrario del objeto SelectCommand que tan sólo interactúa con la fuente de datos para recuperar una porción o todos los datos indicados en el objeto Command anteriormente comentado.

El objeto DataReader

Este objeto es el utilizado en una sola dirección de datos.

Se trata de un objeto de acceso a datos muy rápido.

Este objeto puede usar a su vez el objeto Command o el método ExecuteReader.

5.5.- El concepto DataBinding

DataBinding es una expresión de enlace a datos. Como veremos a continuación es una forma rápida y sencilla de manejar las fuentes de datos mediante su enlace con controles o clases.

El uso de DataBind

El método DataBind se utiliza para rellenar de datos un determinado control o clase. Muchos controles y clases, poseen este método al que le asignaremos un conjunto de datos para que se rellene con ellos, pudiendo después interactuar con los datos de forma directa.

En sí, un DataBinding es un enlace a datos que se encarga de rellenar de datos a un determinado control o clase.

Como ejemplo de esto, veremos como rellenar un control TextBox con un dato utilizando este método.

Iniciaremos una nueva aplicación Windows y escribiremos el siguiente código fuente:

```
using System;
using System.Collections.Generic;
```



```
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

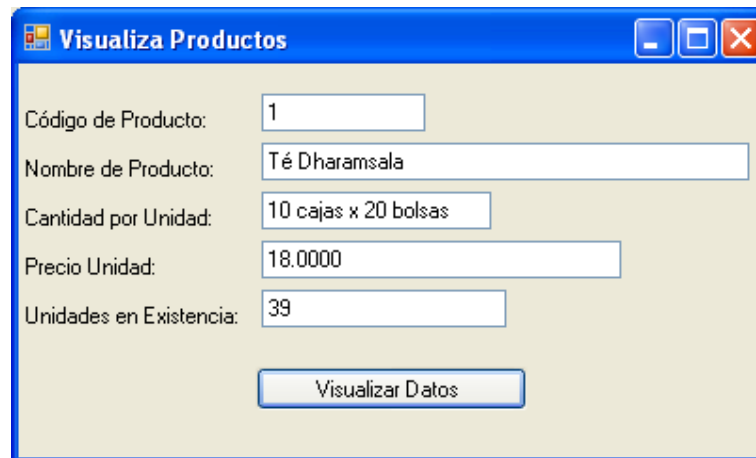
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            private void Form1_Load(object sender, EventArgs e)
            {
            }

            private void button1_Click(object sender, EventArgs e)
            {
                string Conexion = "Data Source=SERVIDOR;Initial
                Catalog=Sistema_Comercial;Integrated Security=True";
                DataSet MiDataSet = new DataSet();
                SqlDataAdapter MiAdaptador = new SqlDataAdapter("SELECT * FROM
                Productos", Conexion);
                MiAdaptador.Fill(MiDataSet,"ListaProductos");

                textBox1.DataBindings.Add("Text", MiDataSet, "ListaProductos.IdProducto");
                textBox2.DataBindings.Add("Text", MiDataSet,
                "ListaProductos.NombreProducto");
                textBox3.DataBindings.Add("Text", MiDataSet,
                "ListaProductos.CantidadPorUnidad");
                textBox4.DataBindings.Add("Text", MiDataSet, "ListaProductos.PrecioUnidad");
                textBox5.DataBindings.Add("Text", MiDataSet,
                "ListaProductos.UnidadesEnExistencia");
            }
        }
    }
}
```

Nuestro ejemplo en ejecución es el que se muestra en la siguiente figura.



Ejemplo en ejecución del uso de DataBinding

Dentro de las conexiones a fuentes de datos, hay algunas partes de éstas que permanecen a veces en el olvido y su importancia sin embargo, es bastante grande.

La acción más pesada cuando realizamos un acceso a una fuente de datos, se encuentra en la conexión con la fuente de datos.

Esa tarea, simple tarea, es la que más recursos del sistema consume cuando accedemos a fuentes de datos.

Esto lo debemos tener en cuenta, y por lo tanto, variante de esto que comentamos son las siguientes premisas:

- La conexión debe realizarse siempre que se pueda, con los proveedores de acceso a datos nativos, que por lo general salvo raras excepciones, serán más rápidos que los accesos a fuentes de datos a través de proveedores del tipo OLE DB y ODBC.
- La conexión con la fuente de datos (apertura de la conexión), debe realizarse lo más tarde posible. Es recomendable definir todas las variables que podamos, antes de realizar la conexión.
- La conexión debe cerrarse lo antes posible, siempre y cuando no tengamos la necesidad de utilizar la conexión previamente abierta.

Hay más particularidades a tener en cuenta cuando trabajamos con fuentes de datos. El hecho de que con un DataSet podamos trabajar con datos desconectados, no significa que dentro de él, podamos abrir una tabla con una cantidad de registros enormes, y trabajemos sobre ella creyendo que esto nos beneficiará.



RESUMEN

ADO .NET permite utilizar tres proveedores de datos. Estos proveedores vinculan las aplicaciones Visual C# a un origen de datos remoto. Y en especial al proveedor de datos de Microsoft SQL Server. Se puede acceder al proveedor de datos a través del espacio de nombres *System.Data.sqlCliente*. Existen seis clases ADO .NET básicas. Estas clases son las clases *Connection*, la clase *Command*, la clase *DataReader*, la clase *DataAdapter*, la clase *DataSet* y la clase *DataView*.

**BIBLIOGRAFÍA RECOMENDADA**

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.

**NEXO**

Esta sección proporciona una descripción de cada una de las clases ADO .NET. centrándose con ejemplos en las propiedades y métodos de las clases que probablemente encuentre con mayor frecuencia en el desarrollo de sistemas.

**ACTIVIDAD**

En la organización que está desarrollando el sistema de información implemente formularios que muestren información de las diferentes tablas que conforman su base de datos.

Poner en práctica todos los programas descritos en la unidad TEMÁTICA.

**AUTOEVALUACIÓN FORMATIVA**

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

- 1.- Diseñar una aplicación basada en Windows que permita autenticar a un usuario del sistema de información. El usuario debe proporcionar su nombre de usuario y su contraseña. Utilice una conexión a una base de datos y la tabla usuarios.
- 2.- Diseñar una aplicación Windows que permita mostrar los datos de una tabla de clientes en controles de cuadros de texto, etiquetas, combos, etc., de acuerdo a las especificaciones de diseño.

Unidad Temática VI

ACCESO CONECTADO A BASE DE DATOS

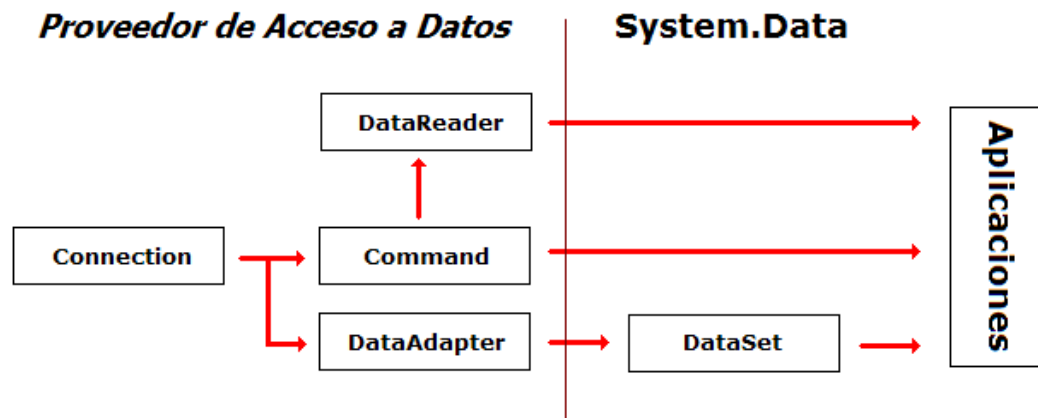
6.1.- El paradigma de la conexión

Cuando abordamos un proyecto de acceso a fuentes de datos, siempre nos encontramos con una duda existencial.

¿Debemos crear una conexión con la base de datos al principio de nuestra aplicación y cerrarla cuando la aplicación se cierre?, ¿o debemos crear una conexión con la base de datos sólo cuando vayamos a trabajar con la fuente de datos?, ¿Y si estamos trabajando continuamente con una fuente de datos?, ¿cómo penalizarían todas estas acciones?

Es difícil de asumir que acción tomar en cada caso, y es que dependiendo de lo que vayamos a realizar, a veces es más efectiva una acción que otra, y en otras ocasiones, no está del todo claro, ya que no existe en sí una regla clara que especifique qué acción tomar en un momento dado.

Lo que sí está claro es que el modelo de datos de ADO.NET que hemos visto, quedaría resumido en cuanto a la conectividad de la manera en la que se representa en la siguiente figura.



Visión general de ADO.NET respecto a la conectividad con bases de datos

El objeto **DataSet** nos ofrece la posibilidad de almacenar datos, tablas y bases de datos de una determinada fuente de datos.

De esta manera, podemos trabajar con las aplicaciones estando desconectados de la fuente de datos.

Sin embargo, a veces necesitamos trabajar con la fuente de datos estando conectados a ella. El objeto **DataReader** nos ofrece precisamente la posibilidad de trabajar con fuentes de datos conectadas.

Por otro lado, el objeto **DataReader** tiene algunas particularidades que conviene conocer y que veremos a continuación.

6.2.- Conociendo el objeto DataReader

El objeto DataReader nos permite como hemos indicado anteriormente, establecer una conexión con una fuente de datos y trabajar con esta fuente de datos sin desconectarnos de ella, sin embargo, hay diferentes cualidades y particularidades que conviene conocer.

DataReader es de solo lectura

Lo que hemos dicho anteriormente, requiere sin embargo, que esta conexión se establezca en un modo de sólo lectura, al contrario de lo que se puede hacer con el objeto DataSet, con el que podemos interactuar con la fuente de datos en modo lectura y modo escritura.

DataReader se maneja en una sola dirección

El objeto DataReader sólo permite que nos desplacemos por los datos en una sola dirección, sin vuelta atrás.

Por el contrario, el objeto DataSet nos permite movernos por los registros para adelante y para atrás.

Además, sólo podemos utilizar el objeto DataReader con conexiones establecidas en una sentencia SQL por ejemplo, pero no podemos variar esta.

Para hacerlo, debemos entonces modificar la conexión con el comando establecido.

DataReader es rápido

Debido a su naturaleza y características, este objeto es bastante rápido a la hora de trabajar con datos.

Como es lógico, consume además menos memoria y recursos que un objeto DataSet por ejemplo.

Sin embargo, dependiendo de las necesidades con las que nos encontremos, puede que este método de acceso y trabajo no sea el más idóneo.

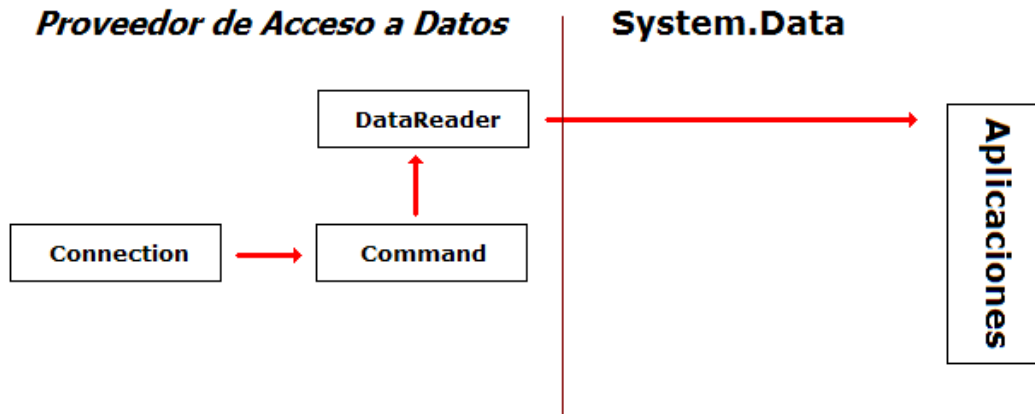
Analizando el flujo de trabajo de DataReader

Cuando trabajamos con fuentes de datos conectadas, trabajaremos con el objeto DataReader.

Para trabajar con este objeto, utilizaremos los objetos siguientes del proveedor de acceso a datos:

- Connection
- Command
- DataReader

Un resumen gráfico de esto es lo que podemos ver en la figura que se muestra a continuación.



El flujo de conectividad de DataReader

6.3.- Un primer contacto con el objeto DataReader

A continuación veremos un ejemplo sencillo sobre la forma de trabajar con DataReader

Un ejemplo simple para entenderlo mejor

Tenga en cuenta que en el siguiente ejemplo nos conectaremos a Microsoft SQL Server y recorreremos los registros uno a uno en un ambiente conectado y volcaremos estos registros en un control TextBox.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

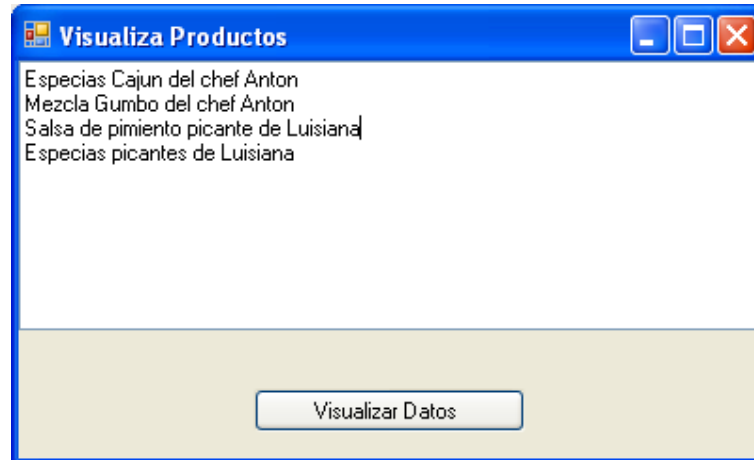
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string CadenaConexion = "Data Source=SERVIDOR;Initial
            Catalog=Sistema_Comercial;Integrated Security=True";
            SqlConnection MiConexion = new SqlConnection(CadenaConexion);
            SqlCommand MiComando = new SqlCommand("SELECT * FROM Productos where
            IdProveedor = 2", MiConexion);
            MiConexion.Open();
            SqlDataReader MiDataReader = MiComando.ExecuteReader();
        }
    }
}
```

```
        while (MiDataReader.Read())  
        {  
            textBox1.Text += MiDataReader["NombreProducto"].ToString()+"\n";  
        }  
        MiConexion.Close();  
    }  
}
```

El código de ejemplo en ejecución es el que se muestra en la siguiente figura.



Ejemplo en ejecución del uso simple de DataReader

Este es un ejemplo simple del uso de DataReader.

Sin embargo, el objeto DataReader contiene un conjunto de propiedades y métodos que nos proporcionan acciones determinadas.

Por ejemplo, en el ejemplo anterior, hemos dado por hecho que la ejecución de la instrucción Select nos devolverá uno o más valores, pero podríamos también saber antes de manipular y trabajar con los posibles datos, si hay o no información.

Esto lo conseguimos con el método HasRows.

6.4.- ¿Trabaja DataReader en un ambiente conectado realmente?

Pese a todo esto, ¿qué ocurre si trabajando en un ambiente conectado se desconecta el servidor de acceso a datos?

Imaginemos por un instante, que la conexión con SQL Server se establece correctamente y que en un momento dado se detiene el servicio del servidor de base de datos. Esto es lo que veremos en el siguiente ejemplo.

Desenchufando la fuente de datos usando DataReader

Inicie un nuevo proyecto, inserte en el formulario de la aplicación un control TextBox y un control Button, y escriba el código que se detalla a continuación:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Data.SqlClient;  
using System.Drawing;  
using System.Text;
```

```
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        SqlConnection MiConexion = new SqlConnection("Data Source=SERVIDOR;Initial
        Catalog=Sistema_Comercial;Integrated Security=True");
        SqlDataReader MiDataReader;
        string strSql = "SELECT * FROM Productos where IdProveedor = 2";
        long contador = 0;
        long posicion = 0;

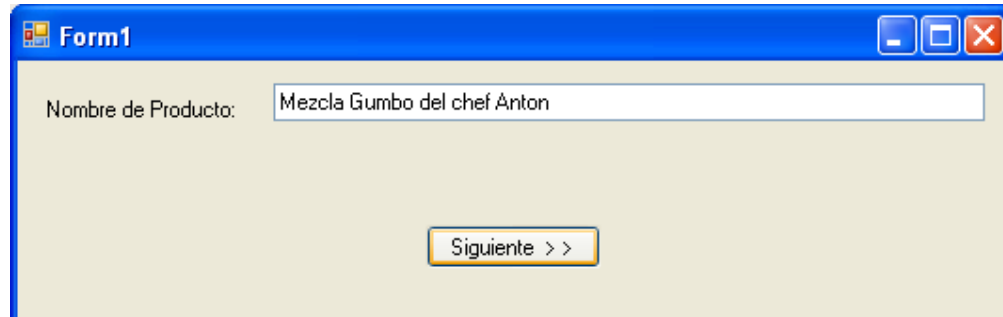
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            Establecer_Conexion(true);
            if (!MiDataReader.HasRows)
                button1.Enabled = false;
            else
                button1_Click(sender, e);
        }

        private void Establecer_Conexion(bool Accion)
        {
            SqlCommand MiComando;
            if (Accion)
            {
                //Creamos el comando
                MiComando = new SqlCommand(strSql, MiConexion);
                //Abrimos la conexion
                MiConexion.Open();
                //Ejecutamos la sentencia SQL
                MiDataReader = MiComando.ExecuteReader();
                //Obtenemos la cantidad de registros obtenidos
                contador = MiDataReader.VisibleFieldCount + 1;
            }
            else
            {
                button1.Enabled = false;
                //Cerramos la conexion
                MiConexion.Close();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            posicion += 1;
            MiDataReader.Read();
            textBox1.Text = MiDataReader["NombreProducto"].ToString();
            //Si hemos recorrido todos los registros finalizamos la conexion
            if (posicion == contador)
                Establecer_Conexion(false);
        }
    }
}
```


Suponiendo que tenemos en nuestra base de datos varios registros, ejecute la aplicación. Si todo ha ido como se esperaba, observaremos que nuestra aplicación tiene un aspecto como el que se muestra en la figura.



Ejemplo en ejecución del uso de DataReader en un ambiente conectado, forzando la desconexión de la fuente de datos

En este punto, detenga el servicio de SQL Server y pulse el botón Siguiente >>.

Observará que la aplicación sigue funcionando.

En este punto se hará la pregunta que todos nos hemos hecho, ¿no es el objeto DataReader un objeto conectado?, ¿cómo es posible que funcione si hemos detenido el servicio de SQL Server?

La respuesta es sencilla.

El objeto DataReader recupera un nutrido conjunto de valores llenando un pequeño buffer de datos e información.

Si el número de registros que hay en el buffer se acaban, el objeto DataReader regresará a la fuente de datos para recuperar más registros.

Si el servicio de SQL Server está detenido en ese momento o en su caso, la fuente de datos está parada, la aplicación generará un error a la hora de leer el siguiente registro.

En sí, DataReader es un objeto conectado, pero trabaja en background con un conjunto de datos, por lo que a veces nos puede resultar chocante su comportamiento como el ejemplo que comento.

6.5.- Usando DataSource con DataReader

¿Podemos usar el método DataSource con el objeto DataReader?.

Demostración del uso de DataSource con DataReader

La respuesta es sí, en ADO.NET 2.0, se ha incorporado un nuevo método al objeto DataTable que le permite tener mayor independencia respecto al modo en el que nos conectemos y recuperemos datos de una fuente de datos.

Recuerde que podemos recuperar datos en modo conectado DataReader o en modo desconectado DataSet.

Este método que se ha incorporado a ADO.NET y que tiene por nombre Load, nos permite cargar un DataReader para volcarlo a continuación dentro de un control como por ejemplo el control DataGridView.

Lo mejor es que veamos como funciona esto con un ejemplo que nos ayude a comprender mejor la teoría.

Inserte en un formulario un control DataGridView y escriba el siguiente código:

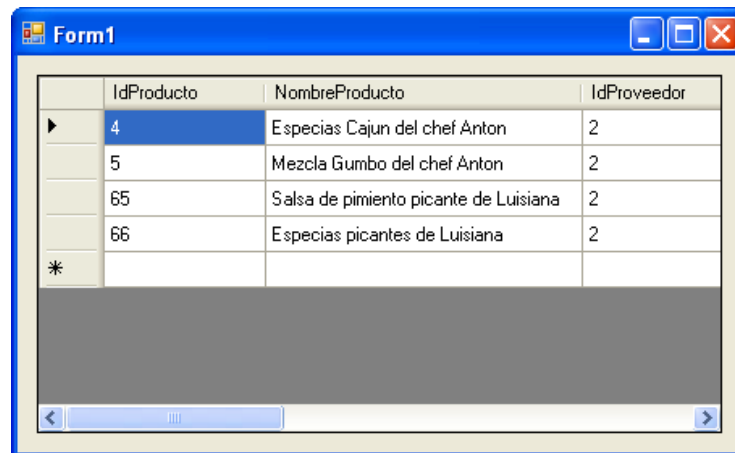
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            SqlConnection MiConexion = new SqlConnection("Data
Source=SERVIDOR;Initial Catalog=Sistema_Comercial;Integrated
Security=True");
            string strSql = "SELECT * FROM Productos where IdProveedor = 2";
            SqlCommand MiComando = new SqlCommand(strSql, MiConexion);
            SqlDataReader MiDataReader;
            DataTable MiTabla = new DataTable();

            //Abrimos la conexion
            MiConexion.Open();
            //Ejecutamos la instruccion SQL
            MiDataReader = MiComando.ExecuteReader();
            //Cargamos en la tabla la lectura del DataReader
            MiTabla.Load(MiDataReader, LoadOption.OverwriteChanges);
            //Volcamos los datos en el dataGridView
            dataGridView1.DataSource = MiTabla;
            //Cerramos la conexion
            MiConexion.Close();
        }
    }
}
```

Nuestro ejemplo en ejecución es el que podemos ver en la siguiente figura.



Ejemplo en ejecución del uso de DataReader y DataSource en un control DataGridView

Con todo y con esto, lo que realmente es curioso, es que hemos olvidado por un instante que el objeto DataReader es un objeto de sólo lectura que funciona en una única dirección, hacia delante.

¿Qué significa esto o como puede influir o como podemos aprovechar esta circunstancia en nuestros desarrollos?

Carga segmentada de datos con DataSource y DataReader

Si recuperamos los datos de una fuente de datos con DataReader y leemos algunos de sus datos y posteriormente, ejecutamos el método DataSource, el resto de datos, aquellos datos que quedan en el DataReader, serán los que se vuelquen en el control que definamos como destino de los datos.

Imaginemos el ejemplo anterior, y el siguiente código fuente.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            SqlConnection MiConexion = new SqlConnection("Data Source=SERVIDOR;Initial
            Catalog=Sistema_Comercial;Integrated Security=True");
            string strSql = "SELECT * FROM Productos where IdProveedor = 2";
            SqlCommand MiComando = new SqlCommand(strSql, MiConexion);
            SqlDataReader MiDataReader;
            DataTable MiTabla = new DataTable();
```

```

//Abrimos la conexion
MiConexion.Open();
//Ejecutamos la instruccion SQL
MiDataReader = MiComando.ExecuteReader();
//Leemos el primer registro por lo que nos posicionamos en el segundo
MiDataReader.Read();
//Cargamos en la tabla la lectura del DataReader
MiTabla.Load(MiDataReader, LoadOption.OverwriteChanges);
//Volcamos los datos en el dataGridView
dataGridView1.DataSource = MiTabla;
//Cerramos la conexion
MiConexion.Close();
}
}
}

```

En este caso, lo que ocurre como ya hemos comentado, es que los datos que se cargan son los que aún no han sido leídos en el objeto DataReader, por lo que se mostrarán todos los datos desde el último leído hasta llegar al final del objeto.

	IdProducto	NombreProducto	IdProveedor
▶	5	Mezcla Gumbo del chef Anton	2
	65	Salsa de pimienta picante de Luisiana	2
	66	Especies picantes de Luisiana	2
*			

6.6.- Usando los componentes de acceso a datos de .NET

Los componentes del entorno .NET nos proporcionan las características necesarias para poder acceder a fuentes de datos de forma rápida y sencilla.

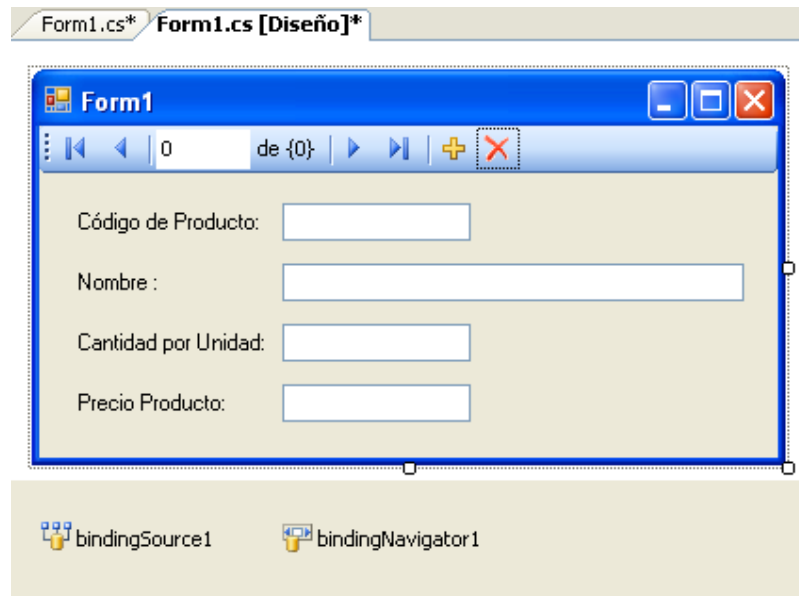
El mejor ejemplo de esto que comento es el que veremos a continuación.

Demostración del uso de BindingSource y BindingNavigator

Para ello, crearemos un proyecto nuevo e insertaremos un control BindingSource y un control BindingNavigator dentro del formulario.

También insertaremos un control TextBox al formulario, donde presentaremos la información sobre la que navegaremos.

Nuestro formulario con los controles insertados en él, tendrá un aspecto similar al que se presenta en la figura.



Controles de navegación y acceso a datos dispuestos en el formulario

Una vez llegado a este punto, lo que tendremos que hacer a continuación será escribir el código fuente necesario para poder representar los datos de la sentencia SQL en el control BindingNavigator, para que a su vez los presente en el control TextBox.

A continuación se indica el código fuente de esta parte de demostración de la aplicación.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            SqlConnection MiConexion = new SqlConnection("Data Source=SERVIDOR;Initial
            Catalog=Sistema_Comercial;Integrated Security=True");
            string strSql = "SELECT * FROM Productos";
            SqlCommand MiComando = new SqlCommand(strSql, MiConexion);
            SqlDataReader MiDataReader;
            DataTable MiTabla = new DataTable();

            //Abrimos la conexion
            MiConexion.Open();
            //Ejecutamos la instruccion SQL
            MiDataReader = MiComando.ExecuteReader();
            //Cargamos en la tabla la lectura del DataReader
        }
    }
}
```

```
MiTabla.Load(MiDataReader, LoadOption.OverwriteChanges);  
//Volcamos los datos en el control TextBox  
bindingSource1.DataSource = MiTabla;  
bindingNavigator1.BindingSource = bindingSource1;  
textBox1.DataBindings.Add("Text", bindingSource1, "IdProducto", true);  
textBox2.DataBindings.Add("Text", bindingSource1, "NombreProducto", true);  
textBox3.DataBindings.Add("Text", bindingSource1, "CantidadPorUnidad", true);  
textBox4.DataBindings.Add("Text", bindingSource1, "PrecioUnidad", true);  
//Cerramos la conexion  
MiConexion.Close();  
}  
}  
}
```



RESUMEN

El objeto DataReader que proporciona Visual Studio .NET permite establecer una conexión con una fuente de datos como por ejemplo Microsoft SQL Server u otras fuentes de datos y trabajar con esta fuente de datos sin desconectarnos de ella, sin embargo, hay diferentes cualidades y particularidades que hacen mas eficiente la programación.



BIBLIOGRAFÍA RECOMENDADA

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.



NEXO

Esta unidad TEMÁTICA proporciona una descripción detallada del objeto DataReader con ejemplos prácticos.



ACTIVIDAD

En la organización que está desarrollando el sistema de información implemente formularios que utilicen DataReader para mostrar información de las diferentes tablas que conforman su base de datos.

Poner en práctica todos los programas descritos en la unidad TEMÁTICA.



AUTOEVALUACIÓN FORMATIVA

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

- 1.- Diseñe una aplicación Windows que permita seleccionar una categoría en un control ComboBox y muestre los productos que forman parte de esa categoría en un control DataGridView.
- 2.- Diseñe una aplicación Windows que permita visualizar imágenes almacenadas en una tabla.

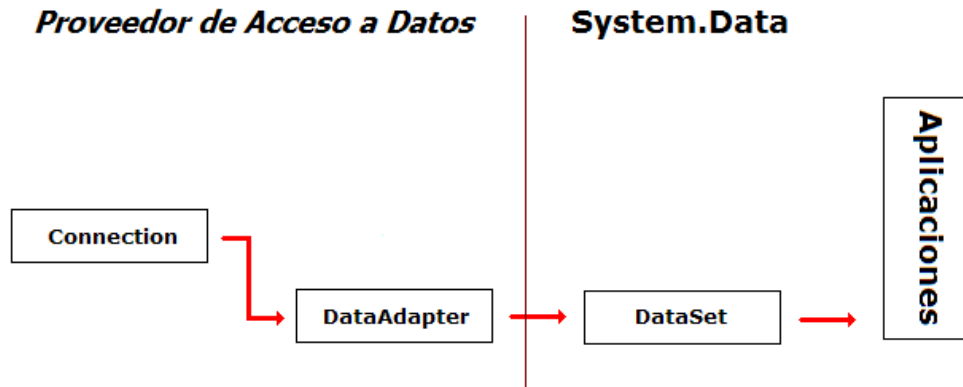
Unidad Temática VII

ACCESO DESCONECTADO A BASE DE DATOS DATASET Y DATAADAPTER

7.1.- Esquema general de la estructura desconectada de acceso a datos

En los capítulos anteriores de este módulo, hemos visto ya el uso de la clase DataSet. Incluso lo hemos visto con algún ejemplo. La clase DataSet está pensada y diseñada para trabajar con fuentes de datos desconectadas. Indudablemente, en este punto, debemos tener clara la estructura general de cómo funciona el acceso desconectado con fuentes de datos.

En la figura mostrada a continuación, podemos observar el diagrama general de esta parte



Estructura general del uso de DataSet en el acceso desconectado a datos

Connection, DataAdapter y DataSet

Como podemos observar en la figura anterior, para comunicarnos con una fuente de datos, siempre deberemos establecer una conexión, independientemente de si la conexión con la fuente de datos va a permanecer a lo largo del tiempo o no.

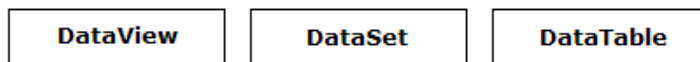
El objeto Connection nos permite por lo tanto, establecer la conexión con la fuente de datos. El objeto DataSet nos permite por otro lado, recoger los datos de la fuente de datos y mandárselos a la aplicación. Entre medias de estos dos objetos, encontramos el objeto DataAdapter que hace las funciones de puente o nexo de unión entre la conexión y el objeto DataSet.

Esto es lo que veremos a continuación, como funciona el objeto DataAdapter, y como encaja todo esto en el acceso a fuentes de datos desconectadas.

7.2.- Conociendo el objeto DataAdapter

El objeto DataAdapter forma parte del proveedor de acceso a datos, tal y como se muestra en la siguiente figura.

System.Data



Proveedor de Acceso a Datos



Visión general de las clases de ADO.NET

Cada proveedor de acceso a datos posee su propio objeto DataAdapter.

Cuando realizamos alguna modificación o acción sobre la fuente de datos, utilizaremos siempre el objeto DataAdapter a caballo entre el objeto DataSet y la fuente de datos establecida a través de la conexión con el objeto Connection.

Con el objeto DataAdapter, podremos además realizar diferentes acciones sobre nuestras bases de datos, acciones como la ejecución general de sentencias de SQL no sólo para seleccionar un conjunto de datos, sino para alterar el contenido de una base de datos o de sus tablas.

Connection, DataAdapter y DataSet

Antes de entrar en materia más profundamente, diremos que en lo que respecta a los proveedores de acceso a datos que vienen integrados con .NET, encontramos dos formas de usar un DataAdapter.

La primera de ellas es utilizando los componentes del proveedor de acceso a datos. La segunda de ellas es utilizando las clases del nombre de espacio del proveedor de acceso a datos.

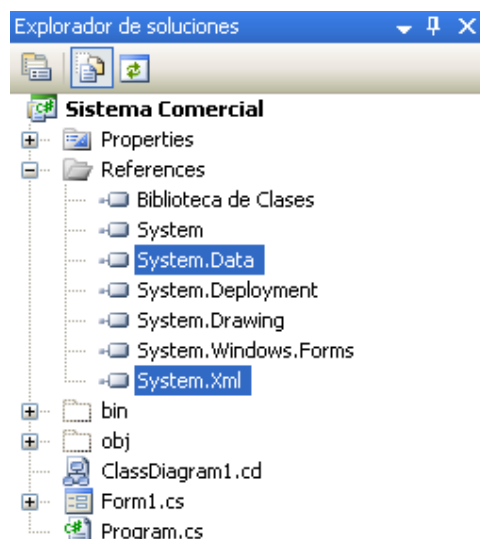
La mejor forma de entender todo esto que comentamos, es trabajando con un ejemplo práctico que nos enseñe a usar el objeto DataAdapter en Visual Studio 2005.

Utilizando las clases de .NET

En este primer ejemplo de demostración del uso de DataAdapter a través de código usando para ello las clases de .NET, estableceremos una conexión con SQL Server y mostraremos los datos recogidos en un control TextBox.

Iniciaremos Visual Studio 2005 y seleccionaremos un proyecto de formulario de Windows. Dentro del formulario, insertaremos un control TextBox y añadiremos dos referencias al proyecto.

Las referencias añadidas serán a las librerías System.Data y System.XML, como se muestra a continuación.



Referencias a las clases de acceso a datos de .NET

Una vez que hemos añadido las referencias necesarias para utilizar las clases que queremos en nuestro proyecto, iremos al código y escribiremos las siguientes instrucciones:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

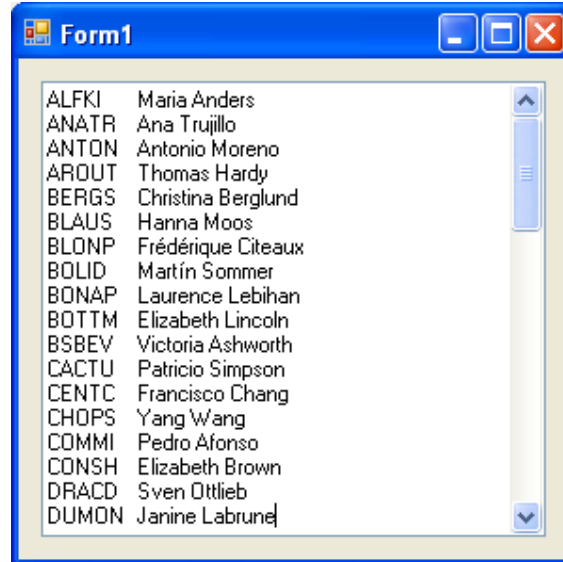
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            SqlConnection MiConexion = new SqlConnection("Data Source=SERVIDOR;Initial
            Catalog=Sistema_Comercial;Integrated Security=True");
            string strSql = "SELECT * FROM Clientes";
            SqlDataAdapter MiAdaptador = new SqlDataAdapter(strSql, MiConexion);
            DataSet MiDataSet = new DataSet();
            MiAdaptador.Fill(MiDataSet, "Clientes");
            //Recorremos todas las filas de la tabla clientes del DataSet
            foreach (DataRow Fila in MiDataSet.Tables[0].Rows)
            {
                textBox1.Text += Fila["IdCliente"].ToString() + "\t" + Fila["NombreContacto"].ToString() +
                "\n";
            }

            MiDataSet = null;
        }
    }
}
```

```
}
```

El ejemplo en ejecución del uso de DataAdapter junto con las clases de .NET es el que se muestra en la figura.



Ejemplo del acceso a datos con DataAdapter a través de las clases de .NET

Utilizando los componentes de .NET

Sin embargo y como ya hemos comentado, existe otro método de acceso a fuentes de datos diferente a las clases de .NET, el acceso a través de componentes que nos faciliten esa tarea. Sin embargo, los componentes de acceso a datos, utilizan por detrás las clases de .NET que hemos visto, lo que ocurre, es que simplifica enormemente el trabajo y ahorra tiempo a la hora de desarrollar aplicaciones.

De todos los modos, todo depende de la utilidad o necesidades con las que nos encontremos en un momento dado.

Iniciaremos un proyecto Windows nuevamente, e insertaremos en él un control TextBox como hicimos en el caso anterior.

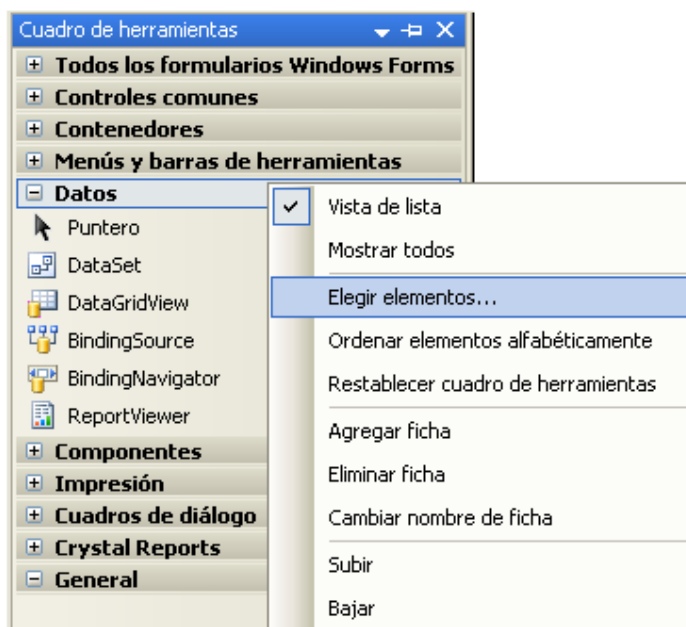
A continuación, añadiremos los componentes de acceso a fuentes de datos SQL Server que es la fuente de datos origen.

Como hemos visto, para conectar a fuentes de datos SQL Server, hemos utilizado el nombre de espacio System.Data y hemos importado en el proyecto los nombres de espacio System.Data y System.Data.SqlClient.

Los componentes .NET de acceso a fuentes de datos de SQL Server, se identifican por el nombre Sqlxxx, siendo xxx el tipo de componente a utilizar.

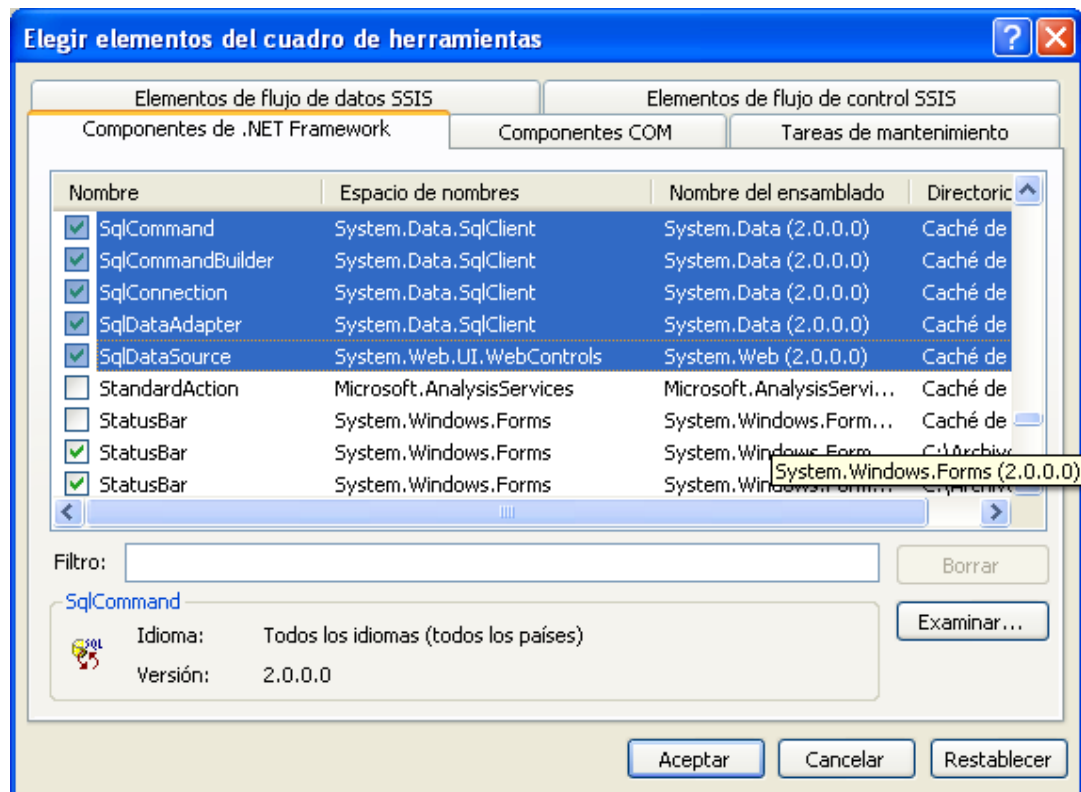
Para poder utilizarlos, deberemos añadirlo a la barra de herramientas.

Para añadir los componentes a nuestro proyecto, haremos doble clic sobre el formulario y posteriormente haremos clic con el botón secundario del mouse sobre la barra de herramientas y seleccionaremos la opción Elegir elementos... (Choose Items...), como se muestra en la siguiente figura.



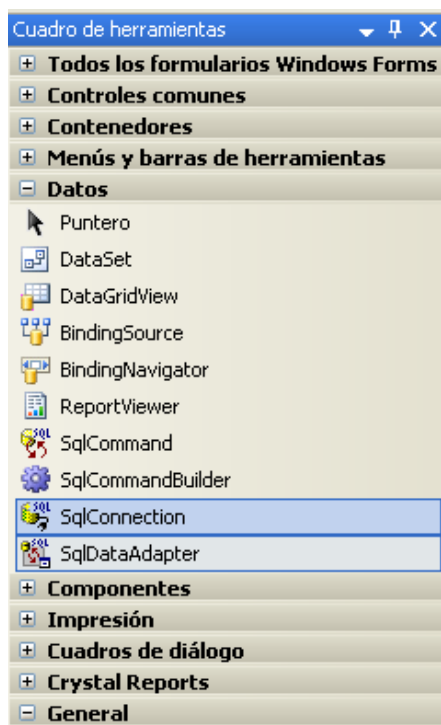
Opción de la barra de herramientas para añadir componentes al entorno

Una vez que hemos hecho esto, seleccionaremos los componentes SqlCommand, SqlCommandBuilder, SqlConnection, SqlDataAdapter y SqlDataSource, tal y como se muestra en la figura mostrada a continuación.



Componentes a añadir al entorno

Una vez que hemos añadido los componentes al entorno, estos quedarán dispuestos dentro de la barra de herramientas.



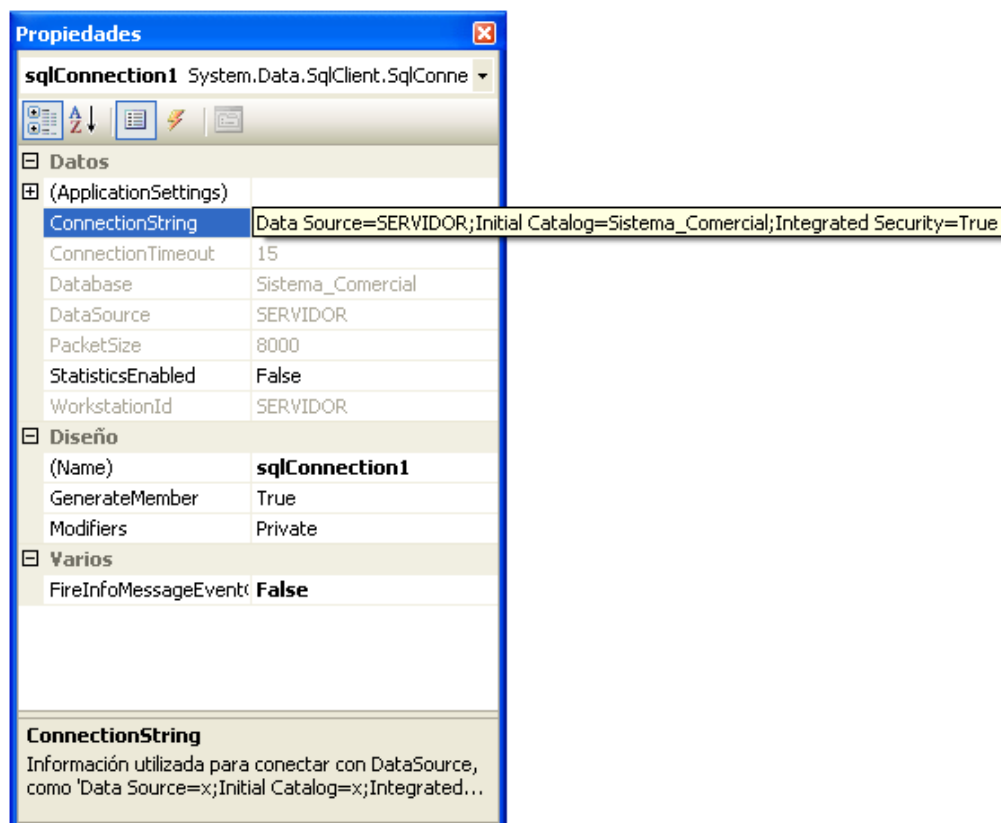
Componentes añadidos en la barra de herramientas

Lo primero que haremos será insertar un componente SqlConnection dentro del formulario.

Acudiremos a la ventana de propiedades del componente y modificaremos la propiedad ConnectionString dentro de la cuál escribiremos la instrucción:

Data Source=SERVIDOR;Initial Catalog=Sistema_Comercial;Integrated Security=Trae

Entendiendo que ésta, es la cadena de conexión válida con nuestra base de datos.



Valores para la propiedad ConnectionString de sqlConnection1

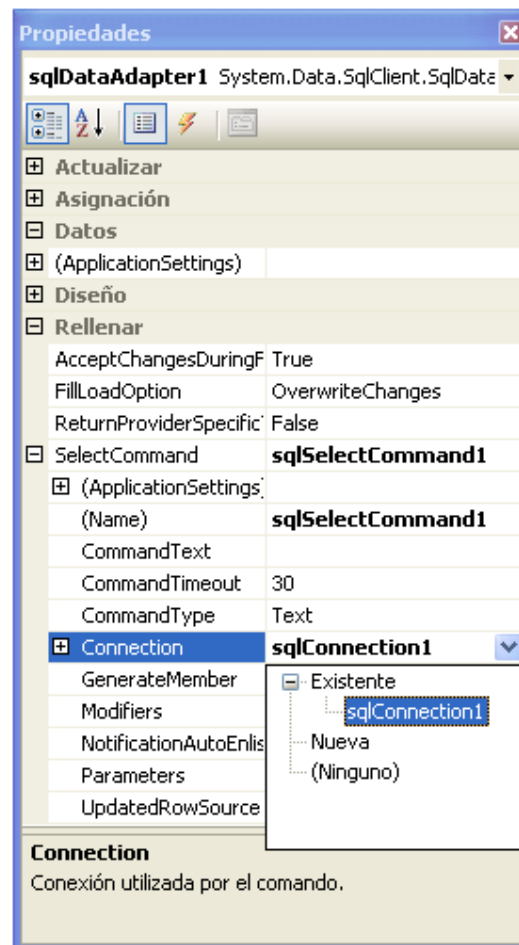
A continuación añadiremos el componente SqlDataAdapter a nuestro formulario.

Si se abre alguna ventana, ciérrela.

Vamos a configurar el control con la ventana Propiedades. Podríamos haberlo hecho desde el asistente que se nos ha abierto, pero lo vamos a hacer de otra forma menos sencilla.

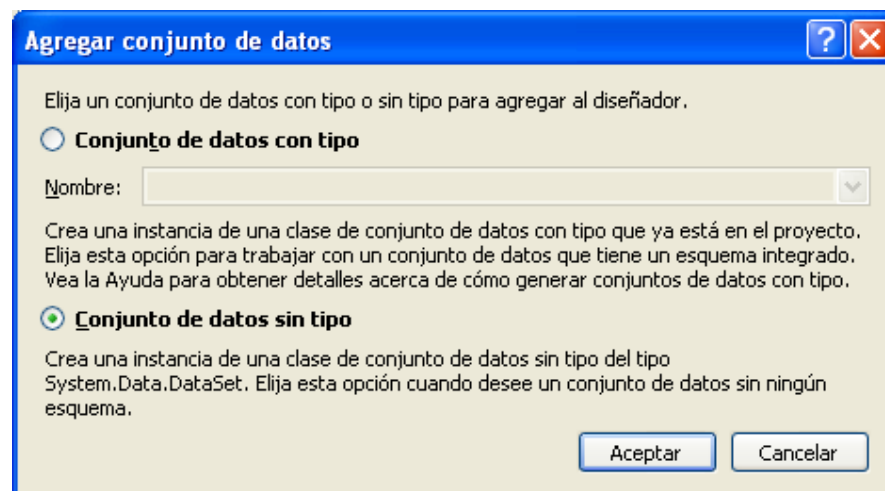
Sitúese sobre la propiedad SelectCommand, y dentro de ésta, en la propiedad Connection. Lo que vamos a hacer, es asignar al componente SqlDataAdapter el componente de conexión que vamos a usar para establecer la comunicación entre la fuente de datos y nuestra aplicación.

Despliegue la propiedad Connection indicada, y seleccione el componente de conexión SqlConnection1 anteriormente configurado, tal y como se muestra en la siguiente figura.

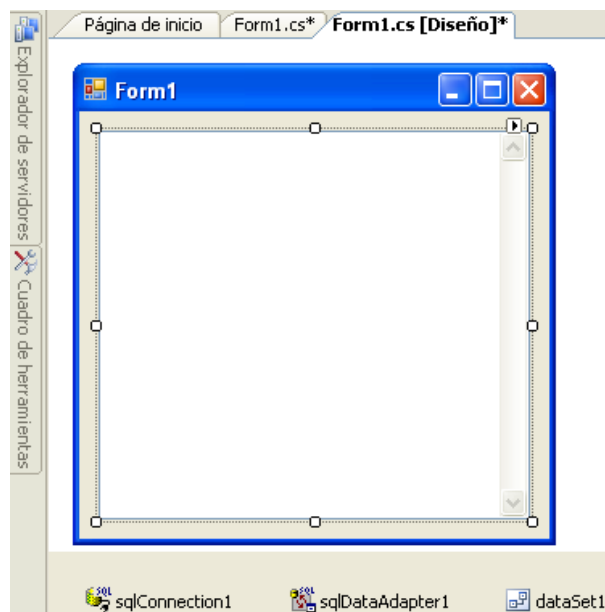


Componente SqlDataAdapter con la conexión establecida para ser usada en la ejecución de nuestra aplicación

Por último, inserte un componente DataSet al formulario.



Todos los componentes quedarán por lo tanto insertados, tal y como se indica en la siguiente figura.



Componentes añadidos en el formulario de nuestra aplicación

Una vez que tenemos todo preparado, tan sólo nos queda escribir la parte de código fuente necesario para poder realizar todas las operaciones y acciones que necesitamos.

A continuación, se expone el código fuente de nuestra aplicación de demostración:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

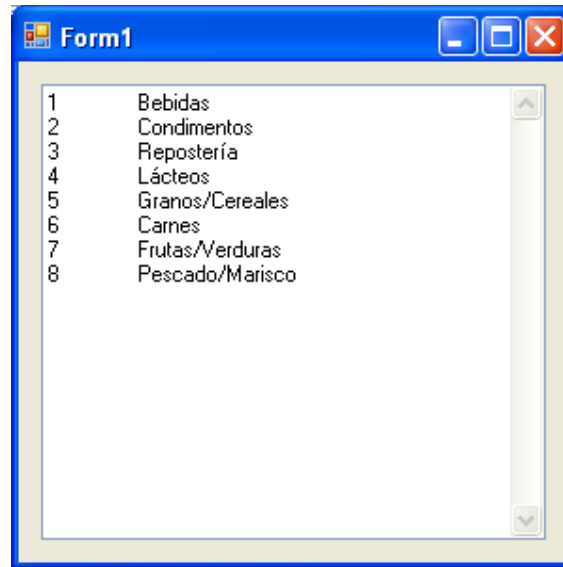
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            sqlDataAdapter1.SelectCommand.CommandText = "SELECT * FROM Categorías";
            //Poblamos el dataSet con los resultados obtenidos de la instruccion SQL
            sqlDataAdapter1.Fill(dataSet1);
            //Recorremos todas las filas de la tabla Categorías del DataSet
            foreach (DataRow Fila in dataSet1.Tables[0].Rows)
            {
                textBox1.Text += Fila["IdCategoria"].ToString() + "\t" +
                    Fila["NombreCategoria"].ToString() + "\n";
            }
        }
    }
}
```



```
}
```

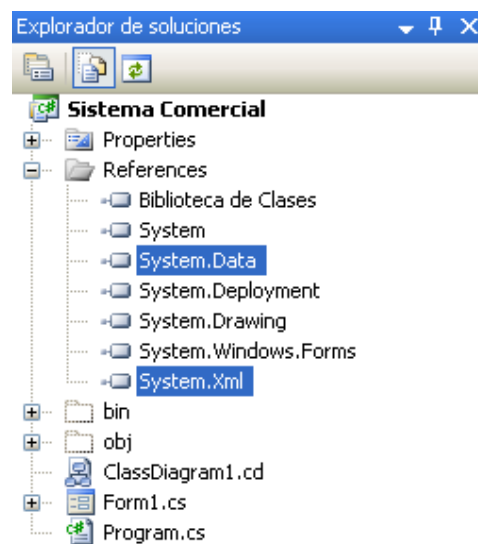
Ahora nos queda únicamente ejecutar nuestra aplicación para estudiar el resultado final.



Ejemplo en ejecución del uso de componentes

Pese a todo lo que hemos visto, quizás se pregunte como es que en el caso del primer ejemplo que hemos visto y en el que hemos declarado el uso de un DataAdapter sin usar componentes, hemos tenido la obligatoriedad de añadir las referencias a los nombres de espacio System.Data y System.Xml, mientras que en este segundo ejemplo, no hemos hecho referencia a ellos.

En realidad nosotros no hemos hecho referencia a ellos, pero al insertar los componentes dentro del formulario, el entorno Visual Studio 2005 se ha encargado por nosotros de añadir esas referencias al proyecto, tal y como puede verse en la figura.



Referencias añadidas automáticamente al trabajar con componentes de acceso a datos

7.3.- Insertando datos a través del objeto DataAdapter



Hasta ahora, todos los ejemplos que hemos visto del objeto `DataAdapter`, han sido ejemplos del uso de selección de datos, pero aún no hemos visto como debemos trabajar cuando realicemos otras acciones sobre la fuente de datos.

A continuación, veremos como realizar acciones de actualización de datos, utilizando para ello el objeto `DataAdapter`.

¿Cómo se insertan datos con el objeto `DataAdapter`

Suponiendo que hemos recogido un conjunto de datos y que trabajando con el objeto `DataSet` hemos realizado una inserción de datos y que a continuación, queremos propagar dicha inserción a la base de datos, deberemos hacer uso del método `Insert` del objeto `DataAdapter`. El objeto `DataAdapter` se encargará de llamar al comando apropiado para cada una de las filas que han sido modificadas en un determinado `DataSet`.

Esto lo realizará siempre a través del método `Update`.

Trabajando con un ejemplo

La mejor manera de ver esto es con un ejemplo que nos ayude a entender mejor como funciona la inserción de datos a través del objeto `DataAdapter`.

Tenga en cuenta además, que la actualización y el borrado de datos funcionan de la misma manera.

Iniciaremos un nuevo proyecto de formulario Windows y en él insertamos los componentes `SqlConnection`, `SqlDataAdapter`, `DataSet` y `SqlCommand`.

Para el componente `SqlConnection`, estableceremos la propiedad `ConnectionString` con el valor:

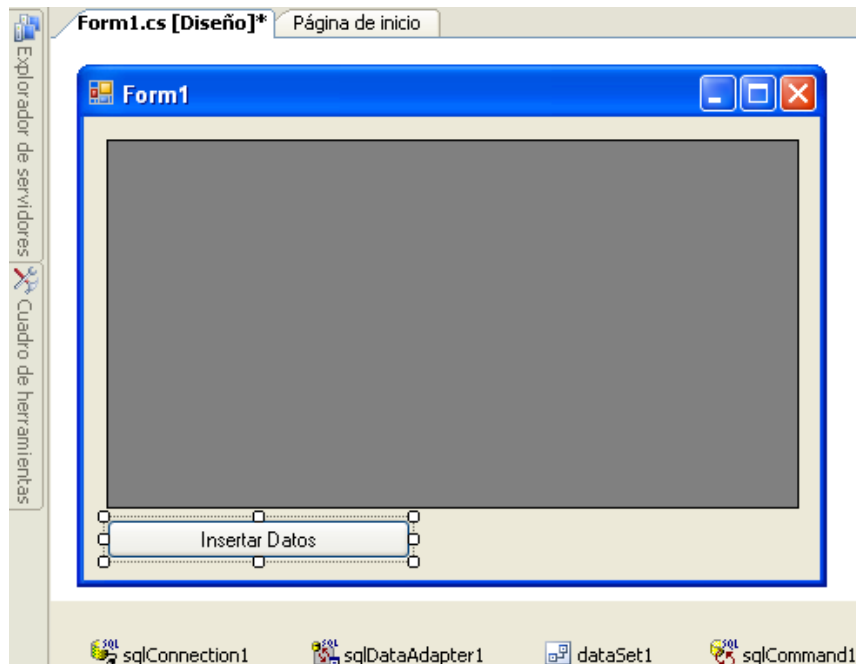
`Data Source=SERVIDOR;Initial Catalog=Sistema_Comercial;Integrated Security=True`

A continuación seleccionaremos el componente `SqlDataAdapter` y modificaremos la propiedad `SelectCommand > Connection` como vimos en el capítulo anterior.

De la lista de posibles conexiones que le aparezca, seleccione la conexión `SqlConnection1`.

Finalmente, inserte un control `Button` y un control `DataGridView` en el formulario.

Éste quedará como se indica en la siguiente figura.



Formulario con los componentes y controles insertados en él

Finalmente, escribiremos el código necesario para ejecutar nuestra aplicación tal y como queremos. Este es el que se detalla a continuación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //Establecemos la cadena SQL a utilizar
            sqlDataAdapter1.SelectCommand.CommandText = "SELECT * FROM Categorias";
            //Poblamos el dataSet con los resultados obtenidos de la instruccion SQL
            sqlDataAdapter1.Fill(dataSet1, "Categorias_Local");
            //Asociamos al control dataGridView1 el DataSet
            dataGridView1.DataSource = dataSet1.Tables["Categorias_Local"];
        }

        private void button1_Click(object sender, EventArgs e)
        {
            DataRow MiFila;
            //Creamos una nueva fila
            MiFila = dataSet1.Tables["Categorias_Local"].NewRow();
            //MiFila["IdCategoria"] = 9; No considerado por ser de tipo identidad
        }
    }
}
```

```

MiFila["NombreCategoria"] = "Cosmeticos";
MiFila["Descripcion"] = "Perfumes";

//Añadimos la fila nueva a la tabla
dataSet1.Tables["Categorias_Local"].Rows.Add(MiFila);
//Miramos si el dataSet ha cambiado
if (dataSet1.HasChanges())
{
    //Creamos el comando de insercion
    sqlCommand1.CommandText = "INSERT INTO Categorias(NombreCategoria,Descripcion)
VALUES(@NombreCategoria,@Descripcion)";
    sqlCommand1.Connection = sqlConnection1;
    sqlDataAdapter1.InsertCommand = sqlCommand1;
    //Añadimos los parametros
    //sqlCommand1.Parameters.Add("@IdCategoria", SqlDbType.Int, 10, "IdCategoria");
    sqlCommand1.Parameters.Add("@NombreCategoria", SqlDbType.NVarChar, 15,
"NombreCategoria");
    sqlCommand1.Parameters.Add("@Descripcion", SqlDbType.NText, 100, "Descripcion");

    sqlConnection1.Open();
    //Actualizamos el dataSet contra la base de datos
    sqlDataAdapter1.Update(dataSet1, "Categorias_Local");
    sqlConnection1.Close();
    MessageBox.Show(" Cambios realizados correctamente !");
}
}
}

```

Por último, ejecute la aplicación. Si todo ha ido correctamente, los datos habrán quedado correctamente insertados en la base de datos.

Un ejemplo de nuestra aplicación en ejecución es la que puede verse a continuación.



Aplicación de ejemplo de inserción de datos con DataAdapter y DataSet en ejecución

Como vemos, el uso de DataAdapter en el caso de manipular datos, varía ligeramente. Sin embargo, no es mucho más complicado con la actualización y borrado de datos. De hecho, la forma de actuar es la misma como veremos a continuación.

7.4.- Actualizando datos a través del objeto DataAdapter

De la misma manera que hemos insertado datos en nuestra base de datos, debemos hacer a la hora de actualizar los mismos.

Sobre la base del ejemplo anterior (componentes y controles), escriba o modifique el siguiente código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            private void Form1_Load(object sender, EventArgs e)
            {
                //Establecemos la cadena SQL a utilizar
                sqlDataAdapter1.SelectCommand.CommandText = "SELECT * FROM Categorias";
                //Poblamos el dataSet con los resultados obtenidos de la instruccion SQL
                sqlDataAdapter1.Fill(dataSet1, "Categorias_Local");
                //Asociamos al control dataGridView1 el DataSet
                dataGridView1.DataSource = dataSet1.Tables["Categorias_Local"];
            }

            private void button1_Click(object sender, EventArgs e)
            {
                //Modificamos la fila 1 columna 4 ( indices comenzando en 0 )
                dataSet1.Tables["Categorias_Local"].Rows[8][2] = "Perfumes y Vanidades";
                //Miramos si el dataSet ha cambiado
                if (dataSet1.HasChanges())
                {
                    //Creamos el comando de actualizacion
                    sqlCommand1.CommandText = "UPDATE Categorias SET Descripcion=@Descripcion
                    WHERE IdCategoria=@IdCategoria";
                    sqlCommand1.Connection = sqlConnection1;
                    sqlDataAdapter1.UpdateCommand = sqlCommand1;
                    //Añadimos los parametros
                    sqlCommand1.Parameters.Add("@IdCategoria", SqlDbType.Int, 10, "IdCategoria");
                    sqlCommand1.Parameters.Add("@NombreCategoria", SqlDbType.NVarChar, 15,
                    "NombreCategoria");
```

```

sqlCommand1.Parameters.Add("@Descripcion", SqlDbType.NText, 100, "Descripcion");
sqlConnection1.Open();
//Actualizamos el dataSet contra la base de datos
sqlDataAdapter1.Update(dataSet1, "Categorias_Local");
sqlConnection1.Close();

MessageBox.Show(" Cambios realizados correctamente !");
    }
}
}

```

Nuestro ejemplo en ejecución es el que se puede ver en la siguiente figura.



Aplicación de ejemplo de actualización de datos con DataAdapter y DataSet

7.5.- Eliminando datos a través del objeto DataAdapter

De igual forma sucede con la eliminación de datos utilizando para ello el objeto DataAdapter junto al objeto DataSet.

Utilizaremos nuevamente en este caso, la base del ejemplo anterior (componentes y controles), y escribiremos el siguiente código:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{

```

```

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        //Establecemos la cadena SQL a utilizar
        sqlDataAdapter1.SelectCommand.CommandText = "SELECT * FROM Categorias";
        //Poblamos el dataSet con los resultados obtenidos de la instruccion SQL
        sqlDataAdapter1.Fill(dataSet1, "Categorias_Local");
        //Asociamos al control dataGridView1 el DataSet
        dataGridView1.DataSource = dataSet1.Tables["Categorias_Local"];
    }

    private void button1_Click(object sender, EventArgs e)
    {
        //Eliminamos la fila 9 (indices comenzando en 0 )
        dataSet1.Tables["Categorias_Local"].Rows[8].Delete();
        //Miramos si el dataSet ha cambiado
        if (dataSet1.HasChanges())
        {
            //Creamos el comando de insercion
            sqlCommand1.CommandText = "DELETE Categorias WHERE
            IdCategoria=@IdCategoria";
            sqlCommand1.Connection = sqlConnection1;
            sqlDataAdapter1.DeleteCommand = sqlCommand1;
            //Añadimos los parametros
            sqlCommand1.Parameters.Add("@IdCategoria", SqlDbType.Int, 10, "IdCategoria");
            sqlCommand1.Parameters.Add("@NombreCategoria", SqlDbType.NVarChar, 15,
            "NombreCategoria");
            sqlCommand1.Parameters.Add("@Descripcion", SqlDbType.NText, 100, "Descripcion");

            sqlConnection1.Open();
            //Actualizamos el dataSet contra la base de datos
            sqlDataAdapter1.Update(dataSet1, "Categorias_Local");
            sqlConnection1.Close();

            MessageBox.Show(" Cambios realizados correctamente !");
        }
    }
}

```

Nuestro ejemplo en ejecución es el que se puede ver en la siguiente figura.



Aplicación de ejemplo de eliminación de datos con DataAdapter y DataSet



RESUMEN

El DataSet permite la conexión a orígenes de datos en un ambiente desconectado, lo más habitual en el desarrollo de sistemas será que nos encontremos trabajando con ambientes y accesos a datos desconectados. Utilizar DataSet conjuntamente con DataAdapter permite sacar el máximo provecho a un ambiente desconectado de datos. Haciendo mas eficiente el desarrollo de sistemas.



BIBLIOGRAFÍA RECOMENDADA

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.



NEXO

Esta unidad TEMÁTICA proporciona una descripción detallada del objeto DataSet y DataAdapter con ejemplos prácticos aplicados a un sistema comercial.



ACTIVIDAD

En la organización que está desarrollando el sistema de información implemente formularios que utilicen DataSet y DataAdapter para mostrar gestionar información de las diferentes tablas que conforman su base de datos (insertar, modificar, actualizar y eliminar datos).

Poner en práctica todos los programas descritos en la unidad TEMÁTICA.



AUTOEVALUACIÓN FORMATIVA

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

1. Diseña una aplicación Windows que muestre un listado de todos los pedidos con

Unidad Temática VIII

ENLACE A FORMULARIOS CON DATASETS TIPADOS

8.1.- ¿Qué son los DataSets tipados?

De forma genérica, podemos definir como DataSet tipado, a aquel DataSet que posee un esquema de datos, a diferencia del DataSet no tipado, que no necesita de ese esquema.

Respecto a los DataSet tipados, diremos que en el entorno de desarrollo, encontramos muchas utilidades y herramientas que nos facilitan el trabajo de este tipo de DataSets. Inclusive, el propio SDK posee herramientas de comandos que nos permite y nos facilita la preparación para trabajar con DataSets tipados.

El esquema al que un DataSet tipado hace referencia, es un documento XML. Se trata de un documento XML con extensión .xsd.

Trabajar con un esquema en lugar de trabajar con una tabla directamente, es mucho más ágil, fácil, rápido y seguro, como veremos más adelante.

Cómo trabajar con un DataSet tipado

Evidentemente, lo que tenemos que tener claro y tomarlo así, como punto de partida, es que si queremos trabajar con un DataSet tipado, tenemos que crear su correspondiente esquema XML.

Esto lo podemos hacer con una herramienta externa, manualmente, o con las herramientas que el entorno de desarrollo de Visual Studio 2005 o el propio SDK nos ofrece.

En nuestro caso, será estas últimas acciones lo que usaremos. Obviamente, hacerlo manualmente es en mi opinión para frikis, máxime cuando sabemos que tenemos herramientas que nos facilitan su creación y nos ahorran mucho tiempo y recursos.

Cuando tengamos ya creado el esquema XML, deberíamos generar la clase del DataSet, algo que a estas alturas, ya lo tenemos dominado.

En uno de los métodos de creación del esquema, el entorno hace ese trabajo por nosotros, en el otro, que es un proceso más manual como veremos, deberíamos crear esa clase con posterioridad, pero aún así, esto último no lo veremos con excesiva profundidad.

¿Qué ventajas nos aportan los DataSets tipados?

Ya hemos enumerado algunas de ellas, rapidez, seguridad, agilidad, facilidad de trabajo, todo ello aplicable a los datos.

Aún así, existen muchas más razones para interesarnos por los DataSets tipados.

A través de los DataSets tipados, podemos realizar acciones que comúnmente son costosas, de manera rápida y sencilla.

Acciones como actualización de datos, modificación, inserción o eliminación de datos, o búsquedas de datos, son algunas de las acciones que en apenas un par de líneas de código, estarán resueltas gracias al uso de DataSet tipados.

Esto lo veremos más adelante con los ejemplos de este capítulo.

Otras acciones adicionales vinculadas con los DataSets tipados son la posibilidad de filtrar datos, ordenar los datos, y trabajar con datos jerárquicos y relaciones de datos.

Una diferencia notable entre los DataSets no tipados y los DataSets tipados, es que los primeros no saben ni tienen conocimiento alguno de lo que almacenan.

Los segundos tienen cierta dosis de inteligencia y conocen el tipo de datos que almacena dentro.

Además de todo esto, el acceso a los datos que guarda un DataSet tipado, así como la manipulación de los mismos, es mucho más simple y requiere menos código, haciendo que el acceso a los datos y el mantenimiento de la aplicación sea más cómoda y sencilla.

8.2.- Generando nuestros DataSets tipados

Dentro de Visual Studio 2005 y de .NET en general, tenemos varias formas de generar nuestros propios DataSets tipados.

Una de las formas de generar DataSets tipados es utilizando el entorno de desarrollo rápido Visual Studio 2005.

Diagrama de datos

Antes de continuar, repasaremos el diagrama de datos con el cuál vamos a trabajar. Este es el que se muestra a continuación.

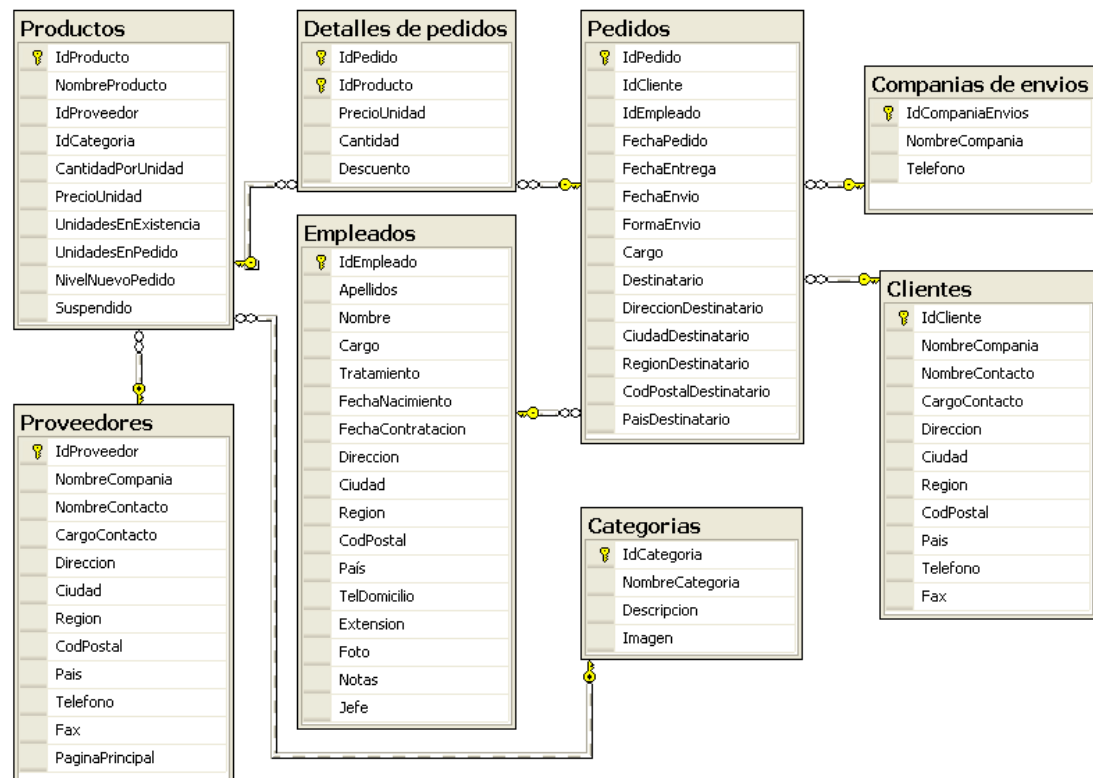


Diagrama de datos con el que vamos a trabajar

Este diagrama nos ayudará a interpretar los datos y a trabajar con ellos, en los ejemplos que veremos a continuación.

A continuación, veremos como generar DataSets tipados desde el entorno de trabajo rápido, es decir Visual Studio 2005.

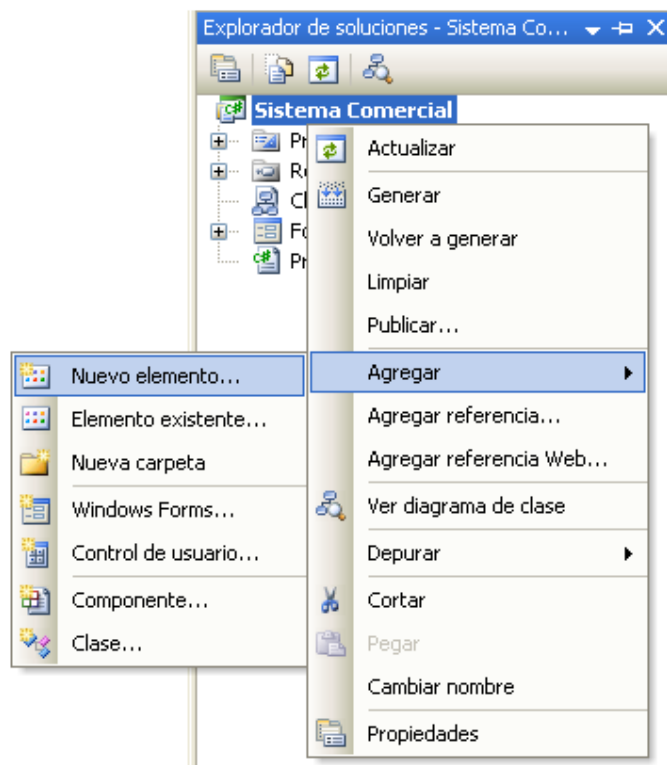
8.3.- Generando un DataSet tipado con Visual Studio 2005

La forma más rápida para generar DataSets tipados es utilizando las herramientas automáticas del entorno Visual Studio 2005.

A continuación veremos como hacer esto de manera rápida y sencilla.

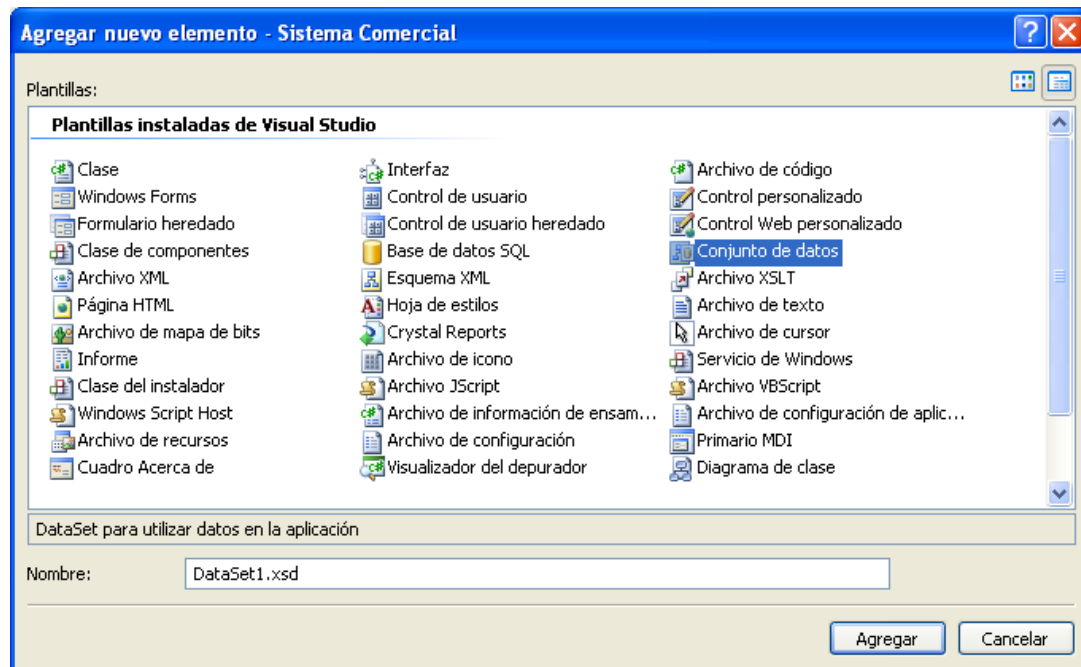
Usando el entorno de desarrollo rápido Visual Studio 2005

Cree un nuevo proyecto de Aplicación para Windows y haga clic con el botón secundario del mouse sobre la ventana Explorador de soluciones, y del menú emergente, seleccione la opción Agregar > Nuevo elemento..., tal y como se muestra en la siguiente figura.



Menú para agregar un elemento al proyecto

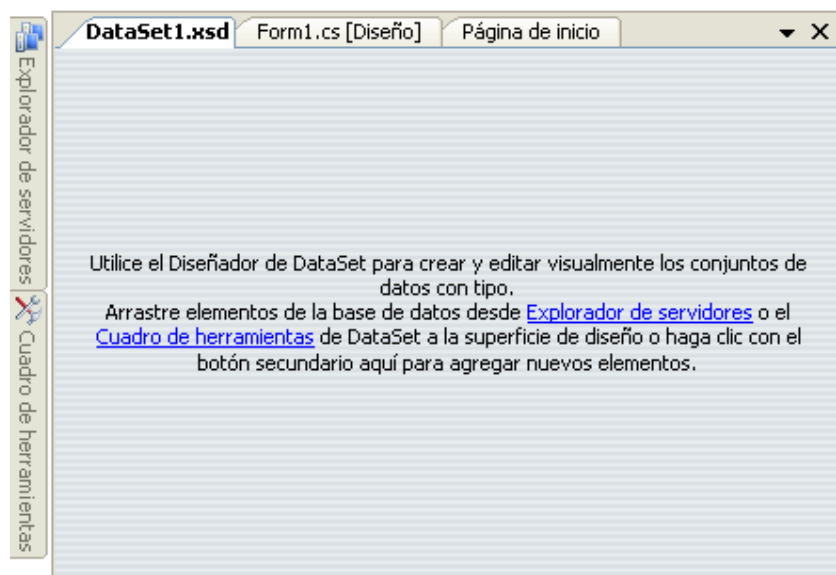
Aparecerá entonces una ventana de Agregar nuevo elemento, dentro de la cuál seleccionaremos la plantilla de Conjunto de datos, tal y como se muestra en la siguiente figura.



Ventana para agregar un nuevo elemento al proyecto

Haga clic en el botón Agregar.

La plantilla del esquema del DataSet, tendrá un aspecto similar al que se muestra en la siguiente figura.

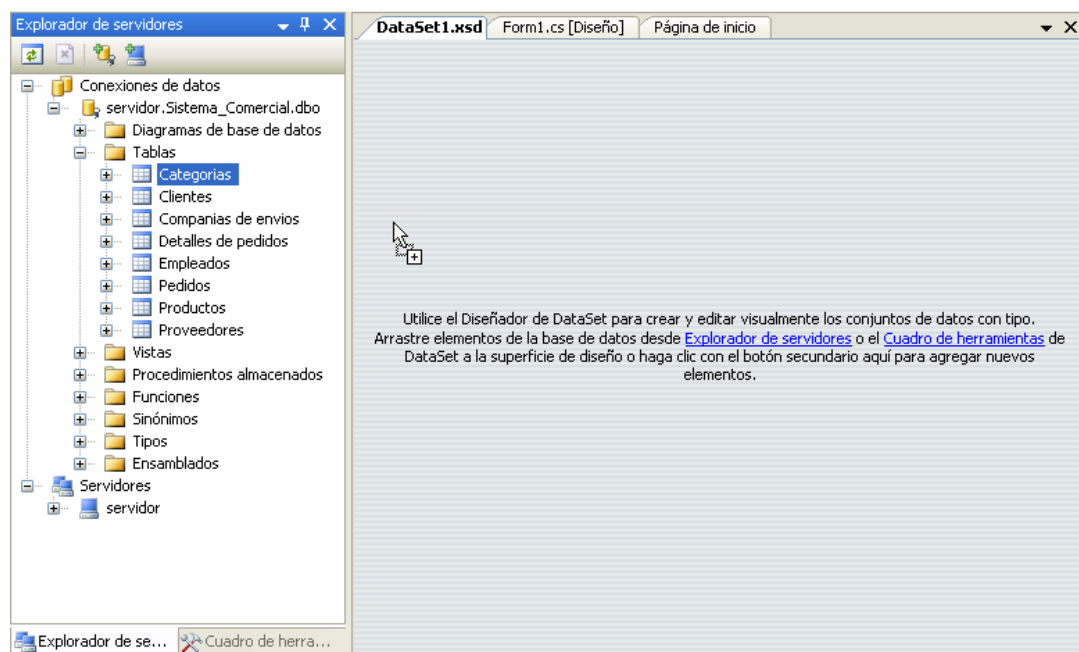


Esquema por defecto del DataSet en Visual Studio 2005

El siguiente paso que deberemos abordar, será la de añadir al esquema, las tablas que queremos que formen parte del DataSet tipado.

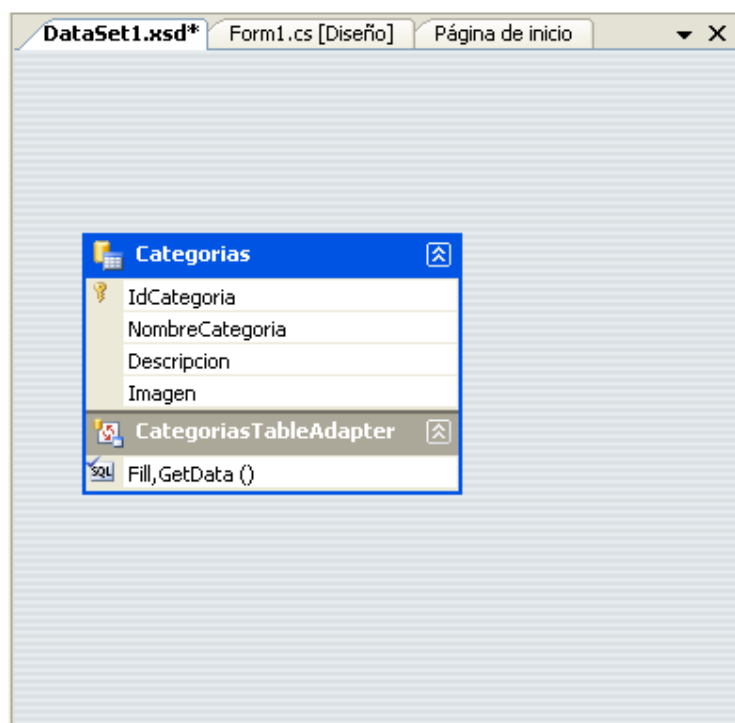
Para hacer esto, abra la ventana Explorador de base de datos y seleccione la base de datos de prueba.

Pulse sobre la tabla Categorías y arrástrela sobre el esquema del DataSet como se indica en la figura



Arrastramos la tabla Categorías sobre el esquema del DataSet


El esquema de la tabla quedará entonces añadido a la aplicación, tal y como se indica en la figura.

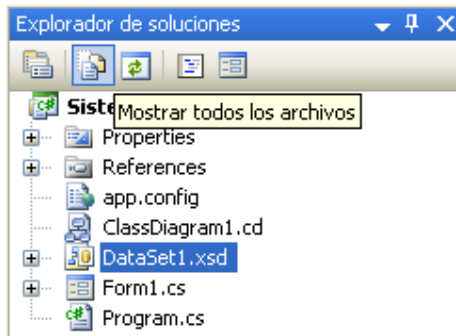


Esquema de la tabla Categorías añadido al entorno de trabajo

En este punto, tendremos listo nuestro DataSet tipado, pero aún no tendremos una clase o código asociado al DataSet tipado.

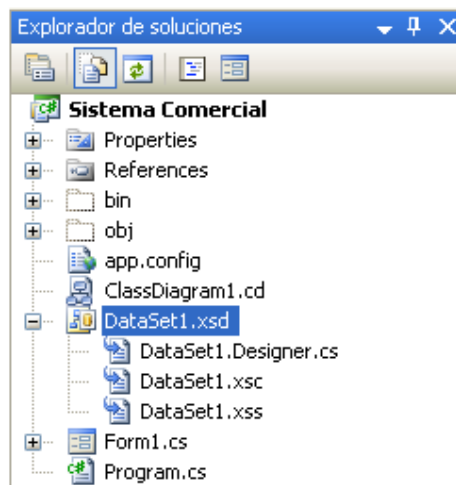
Aún no lo hemos dicho, pero en este punto, Visual Studio 2005 ha generado para nosotros, el código relacionado con el DataSet tipado creado.

Este código, está dentro del directorio de la aplicación, y puede ser accedido a él presionando el botón de la ventana Explorador de soluciones y representado por el siguiente icono , esto es lo que se representa en la siguiente figura.



Opción de mostrar todos los archivos

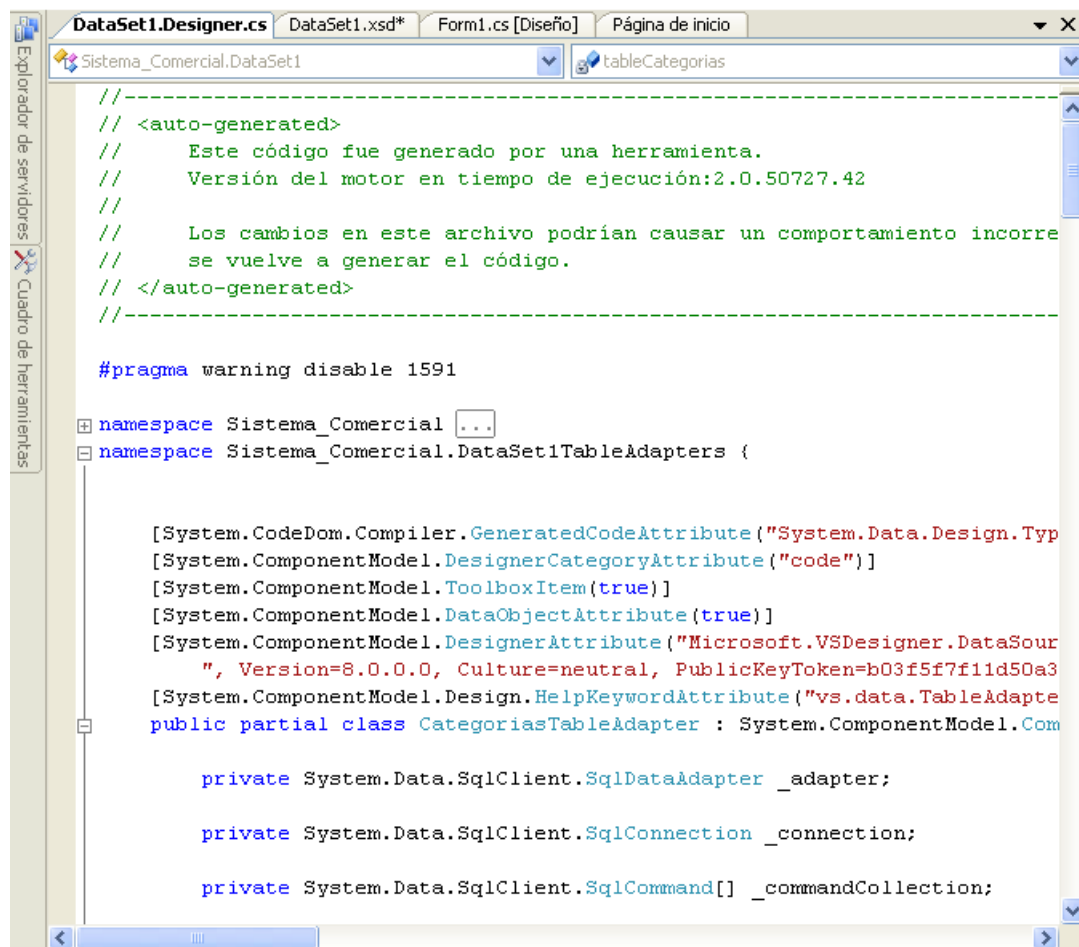
En este punto, veremos que los archivos relacionados con nuestro proyecto y en el caso del elemento DataSet1.xsd, son los que se detallan a continuación.



Archivos del proyecto

Observamos que el archivo DataSet1.Designer.cs depende del archivo DataSet1.xsd, y es el que contiene el código necesario para utilizar el DataSet tipado.

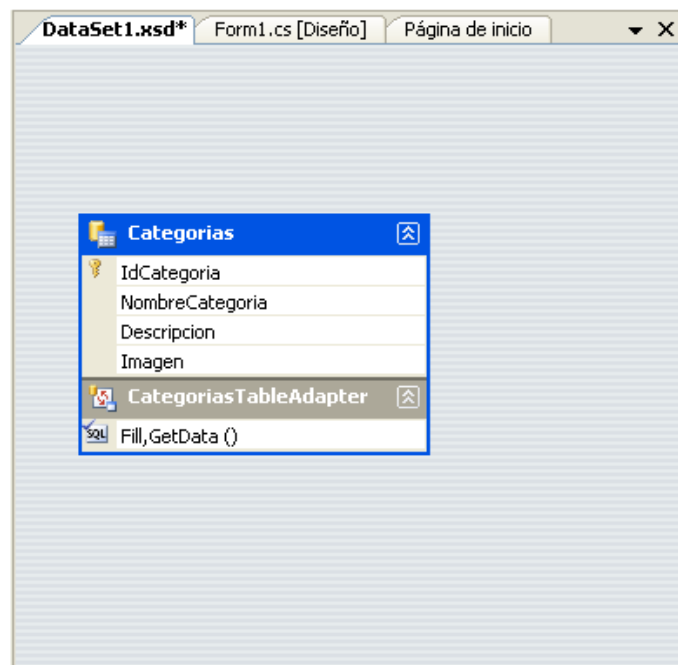
De hecho, si vemos el código generado por Visual Studio 2005, observaremos que coincide con el que se detalla a continuación, visualizamos solo un fragmento:



Segmento de código generado por el DataSet tipado

8.4.- Usando los DataSets tipados

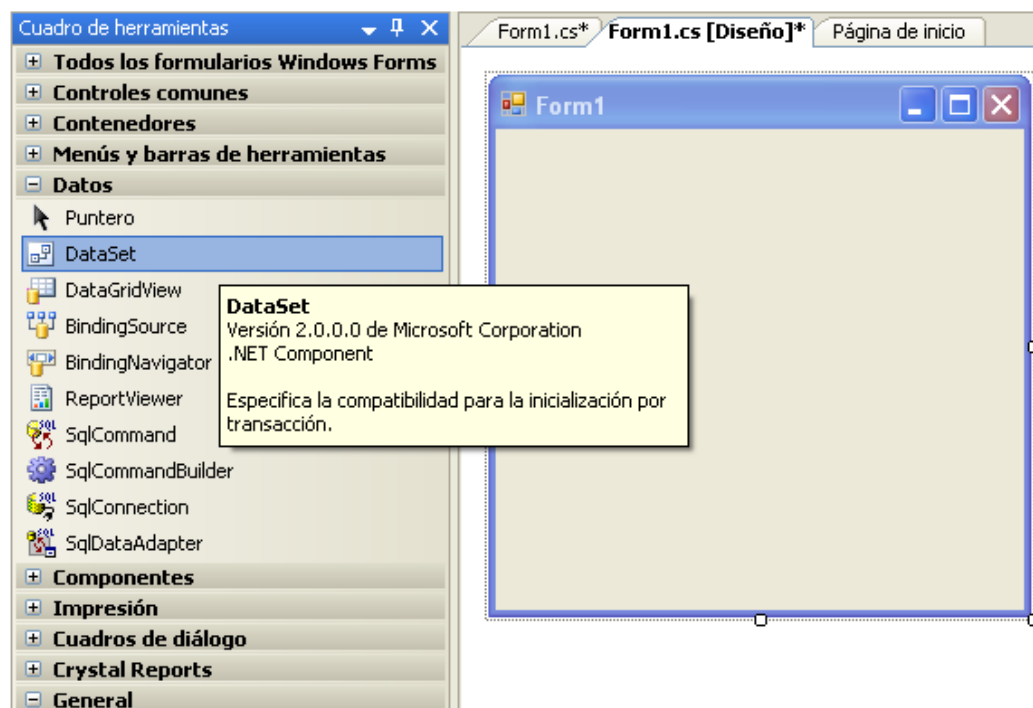
A continuación veremos un ejemplo del uso de DataSets tipados utilizando Visual Studio 2005. Para esto nos servirá lo que ya hemos visto en la generación de un DataSet tipado con Visual Studio 2005. Recordemos que ya habíamos añadido un elemento DataSet1.xsd a nuestro proyecto, tal y como se muestra en la figura.



DataSet1.xsd de ejemplo, añadido al formulario, para utilizarlo como DataSet tipado

Usando las herramientas automáticas para trabajar con DataSets tipados

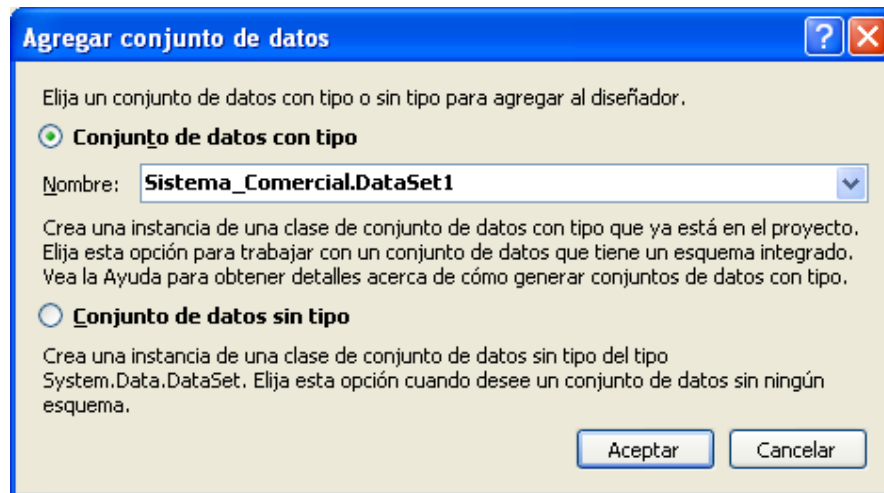
Aún así, vamos a arrastrar sobre el formulario, un componente DataSet como se indica en la siguiente figura.



Selección de un control DataSet del Cuadro de herramientas

En este punto, recordemos que tenemos nuestro DataSet tipado o esquema ya creado y que para usar este esquema desde nuestro objeto DataSet, podemos utilizar las herramientas del entorno .NET.

Cuando arrastramos el componente DataSet sobre el formulario, aparecerá una ventana como que se muestra en la figura, y que nos permitirá indicar si se trata de un DataSet tipado o un DataSet no tipado.

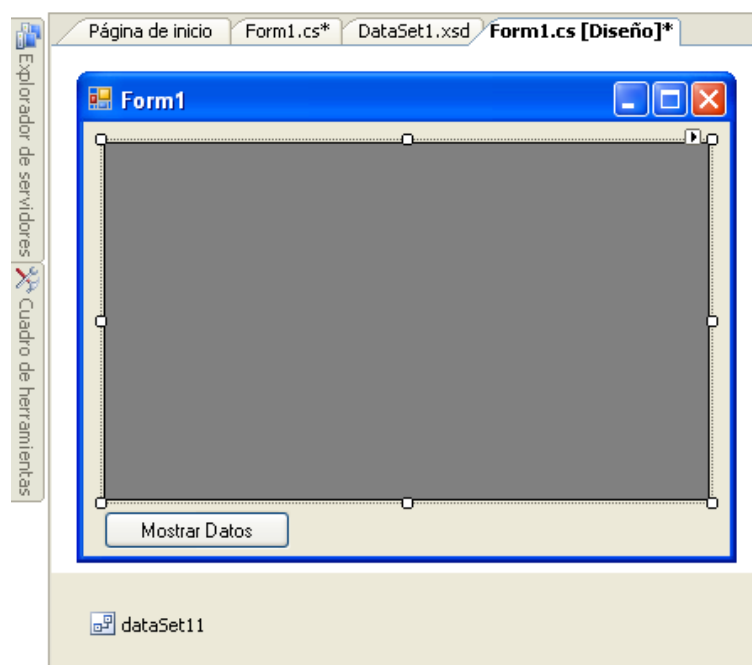


Ventana para agregar un conjunto de datos

Por defecto, aparecerá seleccionada la opción de Conjunto de datos con tipo y el DataSet o esquema que hemos creado.

Presionaremos el botón Aceptar y de esta manera, nuestro objeto DataSet habrá quedado insertado y preparado en el formulario, para utilizar el esquema del DataSet indicado.

El DataSet quedará insertado en nuestro entorno como se indica en la figura.



Control DataSet (dataSet11) insertado en el formulario de ejemplo

Para usar el objeto DataSet insertado, deberemos acceder a él a través de código, de forma muy parecida a lo que lo hacíamos anteriormente.

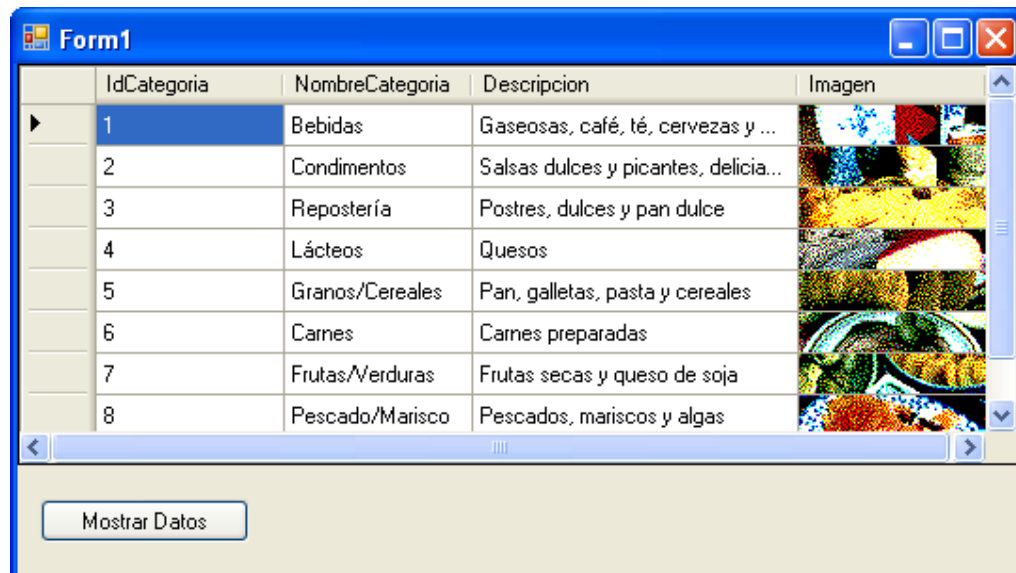
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

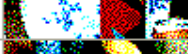
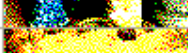
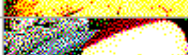
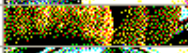
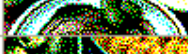
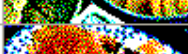
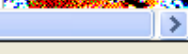
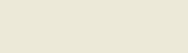
namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Creamos el objeto conexion
            SqlConnection MiConexion = new SqlConnection("Data Source=SERVIDOR;Initial
            Catalog=Sistema_Comercial;Integrated Security=True");
            //Creamos el adaptador para la conexion anterior
            SqlDataAdapter MiAdaptador = new SqlDataAdapter("SELECT * FROM Categorias",
            MiConexion);
            MiAdaptador.Fill(dataSet11, "Categorias");
            //Actualizamos el control dataGridView1
            dataGridView1.DataSource = dataSet11.Categorias;
        }
    }
}
```

El funcionamiento de nuestra aplicación de ejemplo, es igual al que hemos visto en la siguiente figura.



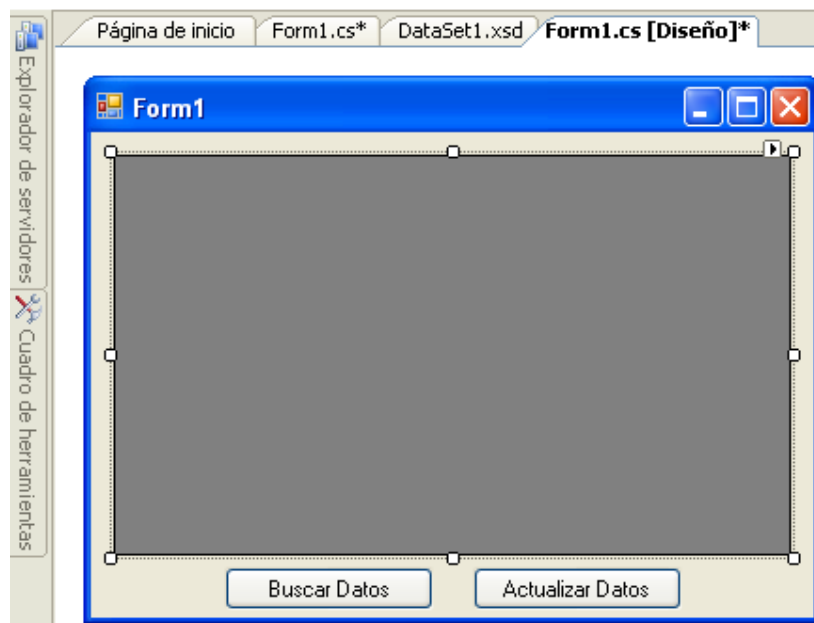
	IdCategoria	NombreCategoria	Descripcion	Imagen
▶	1	Bebidas	Gaseosas, café, té, cervezas y ...	
	2	Condimentos	Salsas dulces y picantes, delicia...	
	3	Repostería	Postres, dulces y pan dulce	
	4	Lácteos	Quesos	
	5	Granos/Cereales	Pan, galletas, pasta y cereales	
	6	Carnes	Carnes preparadas	
	7	Frutas/Verduras	Frutas secas y queso de soja	
	8	Pescado/Marisco	Pescados, mariscos y algas	

Mostrar Datos

Usando DataAdapter con DataSets tipados

Escribiremos a continuación una pequeña aplicación Windows como la hemos hecho anteriormente.

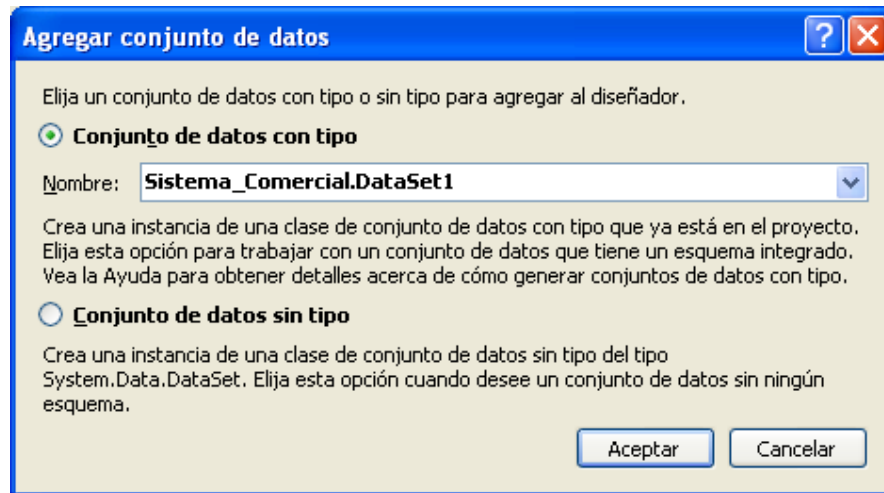
Crearemos nuestro esquema como lo hemos hecho hasta ahora y añadiremos dos botones como se indica en la figura que se muestra a continuación.



Visual Studio IDE showing the design view of 'Form1'. The form contains a large gray rectangular area and two buttons at the bottom: 'Buscar Datos' and 'Actualizar Datos'. The IDE interface includes the 'Explorador de servidores' and 'Cuadro de herramientas' on the left, and tabs for 'Página de inicio', 'Form1.cs*', 'DataSet1.xsd', and 'Form1.cs [Diseño]*' at the top.

Botones del ejemplo añadidos a la ventana Windows

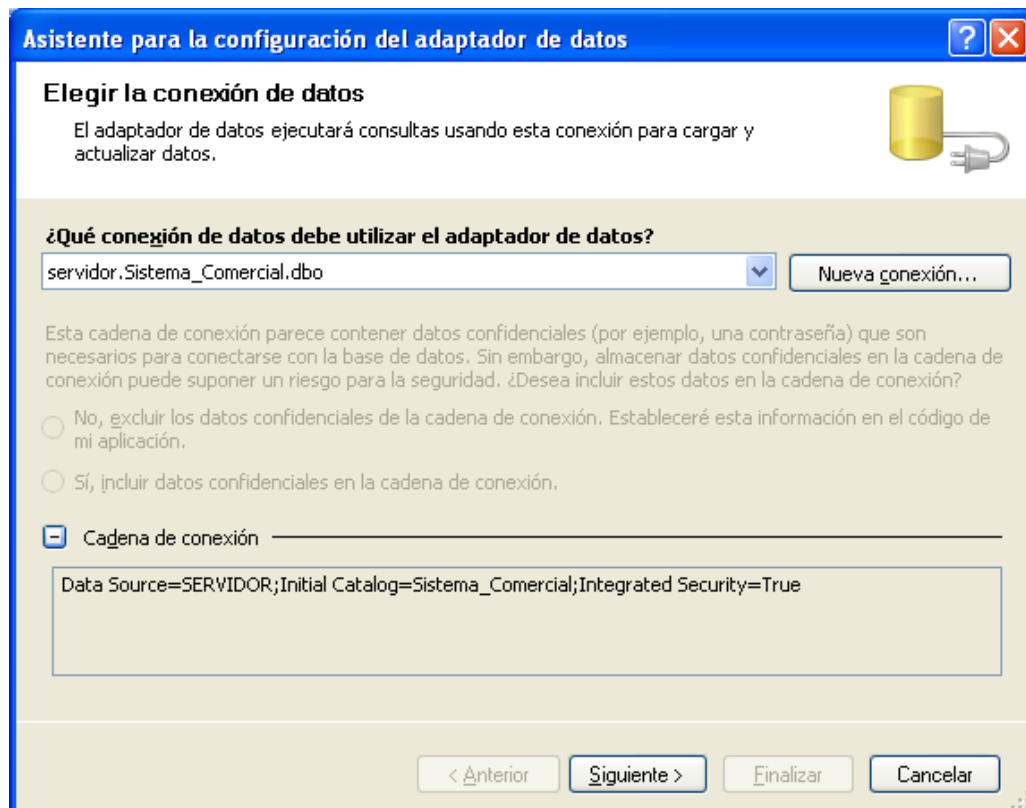
A continuación añadiremos un DataSet al cuál le asignaremos el nombre del DataSet tipado correspondiente al esquema creado, tal y como se indica en la siguiente figura.



DataSet tipado del esquema asignado al objeto DataSet

A este objeto DataSet, le dejamos el nombre que por defecto tiene dataSet11. A continuación, añadiremos un componente SqlDataAdapter al formulario Windows.

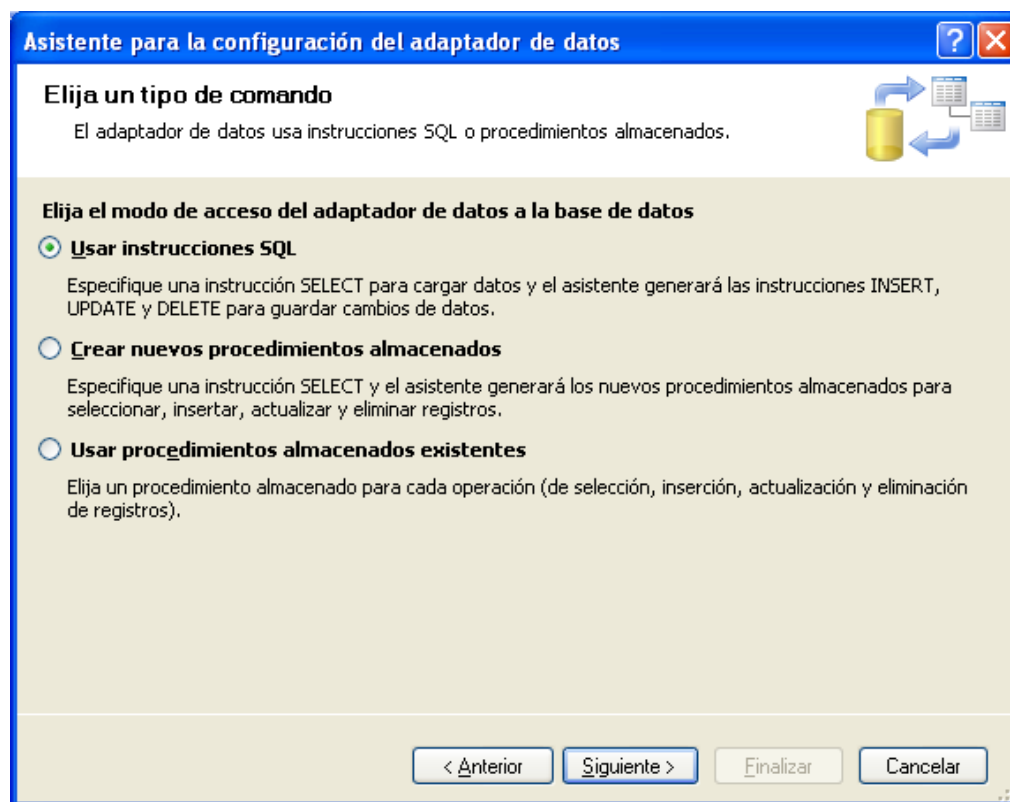
En este punto, aparecerá una ventana como la que se muestra en la figura.



Asistente del SqlDataAdapter

Elegiremos la conexión adecuada y presionaremos el botón Siguiente.

A continuación, el asistente nos mostrará una información como la que se indica en la siguiente figura.

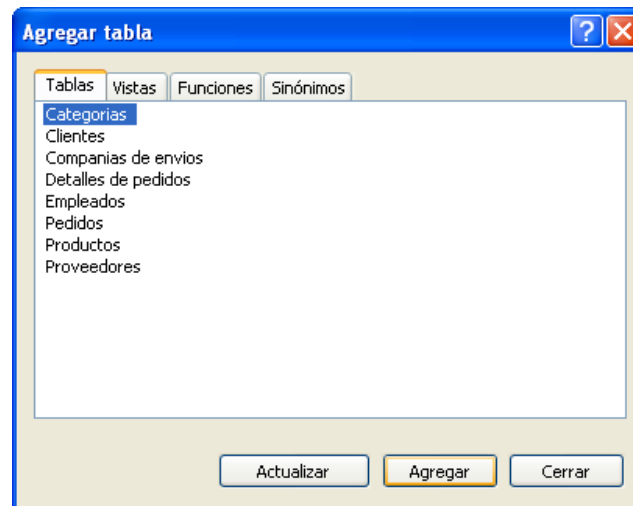


Opción para elegir el tipo de comando a utilizar

Elegiremos la primera de las opciones que es la que se indica en la figura anterior, y presionaremos nuevamente el botón Siguiente.

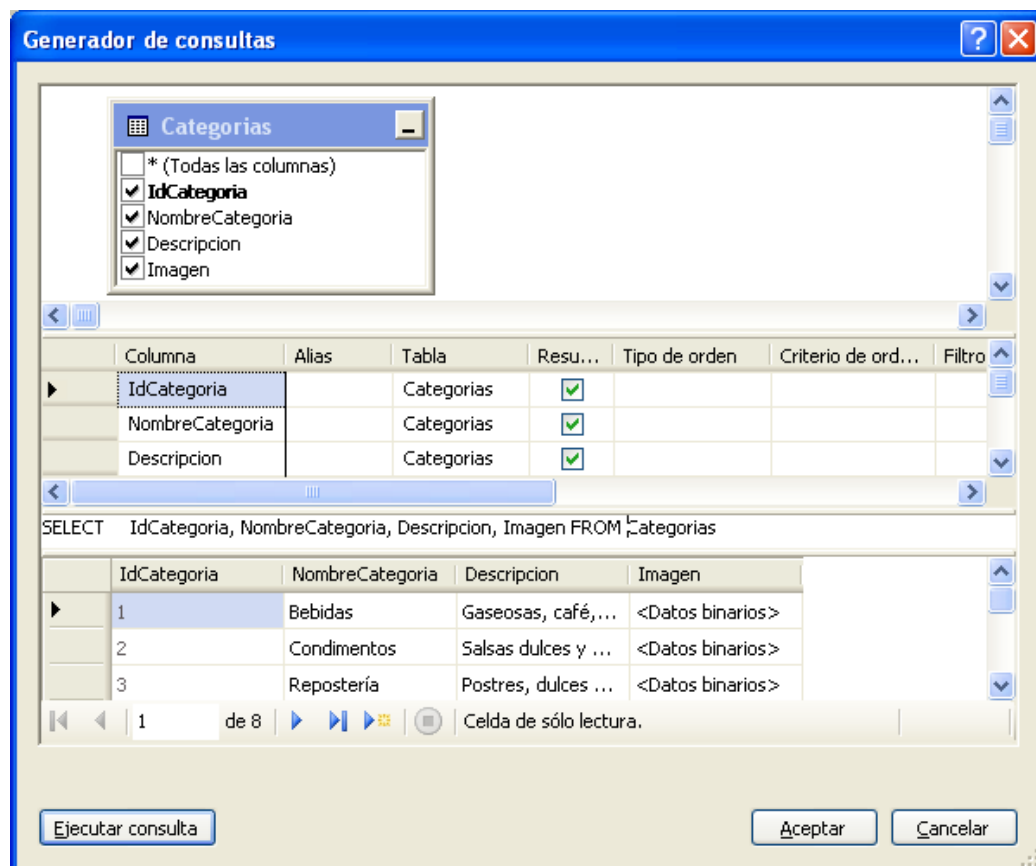
El asistente continuará presentándonos ventanas de información para configurar nuestro componente SqlDataAdapter. Dentro de esta ventana y dentro de los datos que deseamos cargar, escribiremos la instrucción SQL `SELECT * FROM Categorías` o utilizar el generador de consultas.

Mediante el generador de consultas lo primero que debe hacer es agregar las tablas que desea incluir en la consulta. Para nuestro ejemplo utilizaremos la tabla categorías.



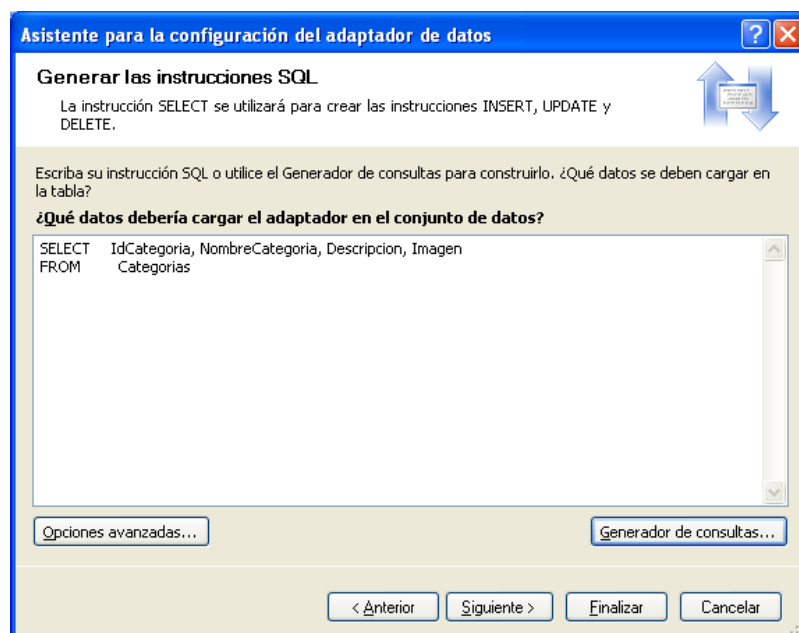
Ventana agregar tabla

Seleccione los campos que desea obtener y ejecute la consulta para visualizar los resultados, Acepte para finalizar con el generador de consultas.



Ventana generador de consultas

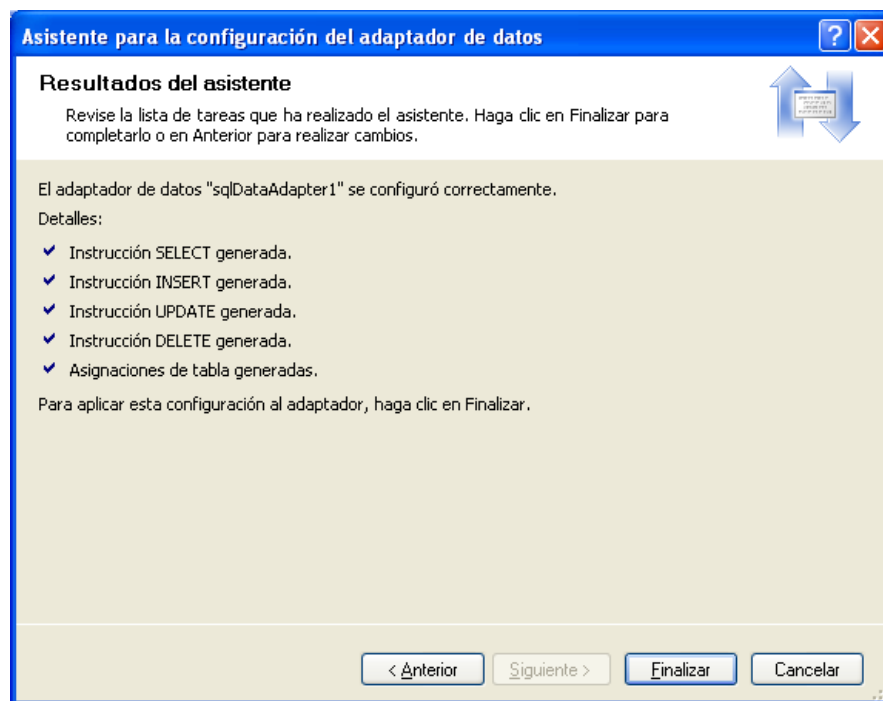
El código generado será insertado en la ventana de configuración del adaptador de datos.



Opción de generación de las instrucciones SQL

A continuación, presionaremos el botón Siguiente.

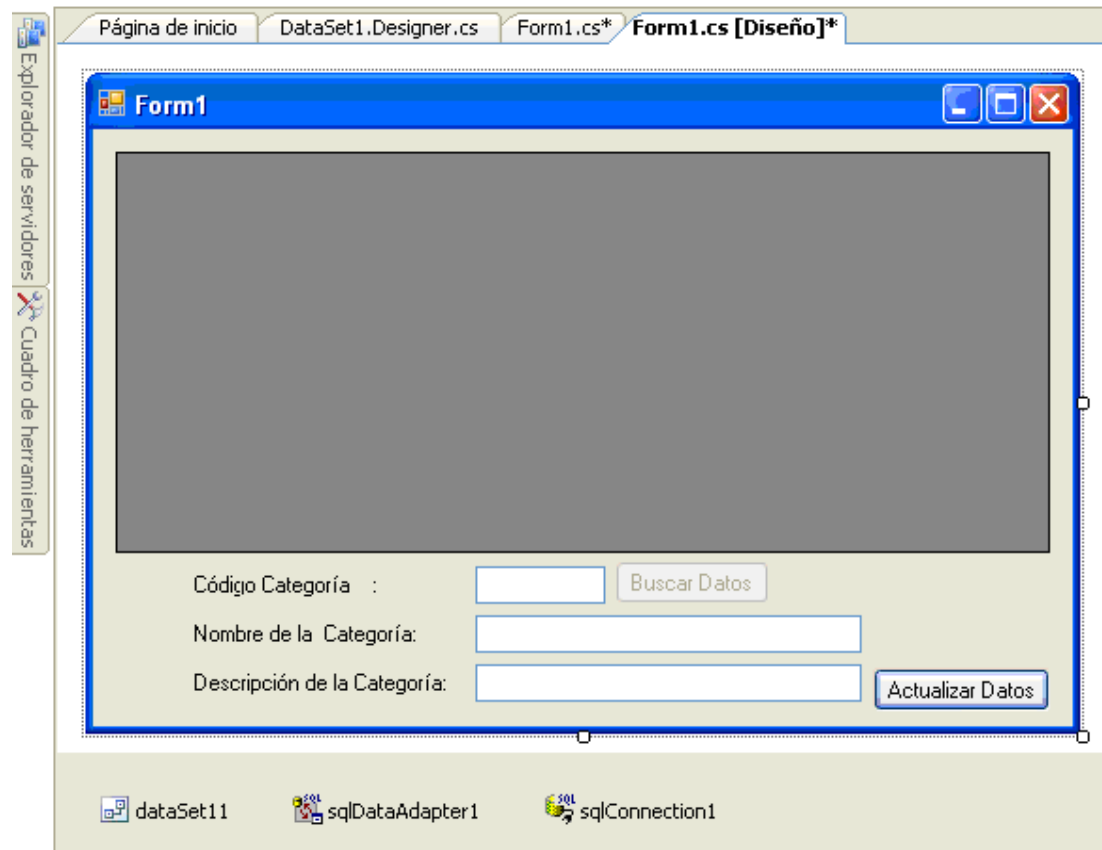
De esta manera, el asistente terminará de crear las instrucciones necesarias para trabajar con el componente SqlDataAdapter. Si todo ha ido correctamente, aparecerá una ventana como la que se indica en la siguiente figura.



Resultados de la configuración del componente SqlDataAdapter

Para concluir, haremos clic sobre el botón Finalizar.

Automáticamente, en Visual Studio 2005, aparecerá añadido al formulario Windows el componente SqlConnection, como vemos en la siguiente figura.



Componentes añadidos al formulario Windows

A continuación, escribiremos el código fuente de la aplicación, que se encarga de recoger los datos de la base de datos, insertarlos en un DataSet, para modificarlos y actualizar las modificaciones en la base de datos.

El código fuente de la aplicación, quedará como se detalla a continuación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {

```

```

        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        sqlDataAdapter1.Fill(dataSet11, "Categorias");
        dataGridView1.DataSource = dataSet11.Categorias;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        int cod_busqueda = Convert.ToInt32(textBox1.Text);
        DataSet1.CategoriasRow MisDatos =
        dataSet11.Categorias.FindByIdCategoria(cod_busqueda);

        //visualizamos los datos
        textBox2.Text = MisDatos.NombreCategoria;
        textBox3.Text = MisDatos.Descripcion;
        //Cambiamos los datos para el registro con IdCategoria = 4
        MisDatos.NombreCategoria = "Lacteos";
        MisDatos.Descripcion = "Quesos";
        //Deshabilitamos como medida de seguridad el botón
        button1.Enabled = false;
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (dataSet11.HasChanges())
        {
            //Declaramos un DataSet que únicamente contiene los cambios
            DataSet dataModificado = dataSet11.GetChanges();
            //Actualizamos los cambios del data set mediante el adaptador
            sqlDataAdapter1.Update(dataModificado);
            //Aceptamos los cambios en el dataset para seguir trabajando con el
            dataSet11.AcceptChanges();
            MessageBox.Show("Los cambios del DataSet han sido actualizados en la base de datos");
        }
        else
        {
            MessageBox.Show("NO hay cambios en el DataSet");
        }
    }
}

```

Nuestro ejemplo en ejecución, es el que puede verse a continuación.

The screenshot shows a Windows application window titled "Form1". It contains a data grid with the following columns: IdCategoria, NombreCategoria, Descripción, and Imagen. The grid has 8 rows of data. Below the grid, there are input fields and buttons:

IdCategoria	NombreCategoria	Descripción	Imagen
1	Bebidas	Gaseosas, café, t...	[Image]
2	Condimentos	Salsas dulces y p...	[Image]
3	Repostería	Postres, dulces y ...	[Image]
4	Lacteos	Quesos	[Image]
5	Granos/Cereales	Pan, galletas, pa...	[Image]
6	Carnes	Carnes preparadas	[Image]
7	Frutas/Verduras	Frutas secas y qu...	[Image]
8	Pescado/Marisco	Pescados, marisc...	[Image]

Below the grid, there is a text box labeled "Ingrese Código Categoría:" with the value "4" entered. To its right is a button labeled "Buscar Datos". Below that, there are two more text boxes: "Modifique Nombre de la Categoría:" with the value "Lacteos" and "Modifique Descripción de la Categoría:" with the value "Quesos". To the right of the second text box is a button labeled "Actualizar Datos".

Below the main form, there is a small dialog box with a blue title bar and a red close button. It contains the text "Los cambios del DataSet han sido actualizados en la base de datos" and an "Aceptar" button.

Ejemplo del DataAdapter y DataSet tipado en ejecución

Para insertar registros, diseñe otra interfaz y añada los controles necesarios para obtener la interfaz que se muestra en la siguiente figura.

The screenshot shows a Windows application window titled "Form1" in design mode. The window contains three text boxes and one button:

Código Categoría Generado: [Text Box]

Ingrese Nombre de la Categoría: [Text Box]

Ingrese Descripción de la Categoría: [Text Box]

Insertar Registro [Button]

At the bottom of the window, there are three icons representing data components: dataSet11, sqlDataAdapter1, and sqlConnection1.

Ejemplo del DataAdapter y DataSet tipado en ejecución

El botón insertar registro nos servirá para añadir una fila nueva a nuestra base de datos.

El código fuente de nuestra aplicación de ejemplo, es el que se detalla a continuación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            sqlDataAdapter1.Fill(dataSet11, "Categorias");
            dataGridView1.DataSource = dataSet11.Categorias;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Añadimos una nueva fila al dataSet tipado
            dataSet11.Categorias.AddCategoriasRow(textBox2.Text, textBox3.Text);
            //Actualizamos el dataSet
            sqlDataAdapter1.Update(dataSet11);
            MessageBox.Show(" Fila añadida correctamente ");
        }
    }
}
```

Los resultados se muestran en el gráfico que se muestra a continuación.

The screenshot shows a Windows form titled 'Form1' with a table containing category data. Below the table are input fields for 'Código Categoría Generado:', 'Ingrese Nombre de la Categoría:', and 'Ingrese Descripción de la Categoría:', along with an 'Insertar Registro' button. A message box is displayed below the form, stating 'Fila añadida correctamente' with an 'Aceptar' button.

	IdCategoría	NombreCategoría	Descripción	Imagen
	2	Condimentos	Salsas dulces y p...	
	3	Repostería	Postres, dulces y ...	
	4	Lacteos	Quesos	
	5	Granos/Cereales	Pan, galletas, pa...	
	6	Carnes	Carnes preparadas	
	7	Frutas/Verduras	Frutas secas y qu...	
	8	Pescado/Marisco	Pescados, marisc...	
	9	Perfumes	Variedades	
*				

Código Categoría Generado:

Ingrese Nombre de la Categoría:

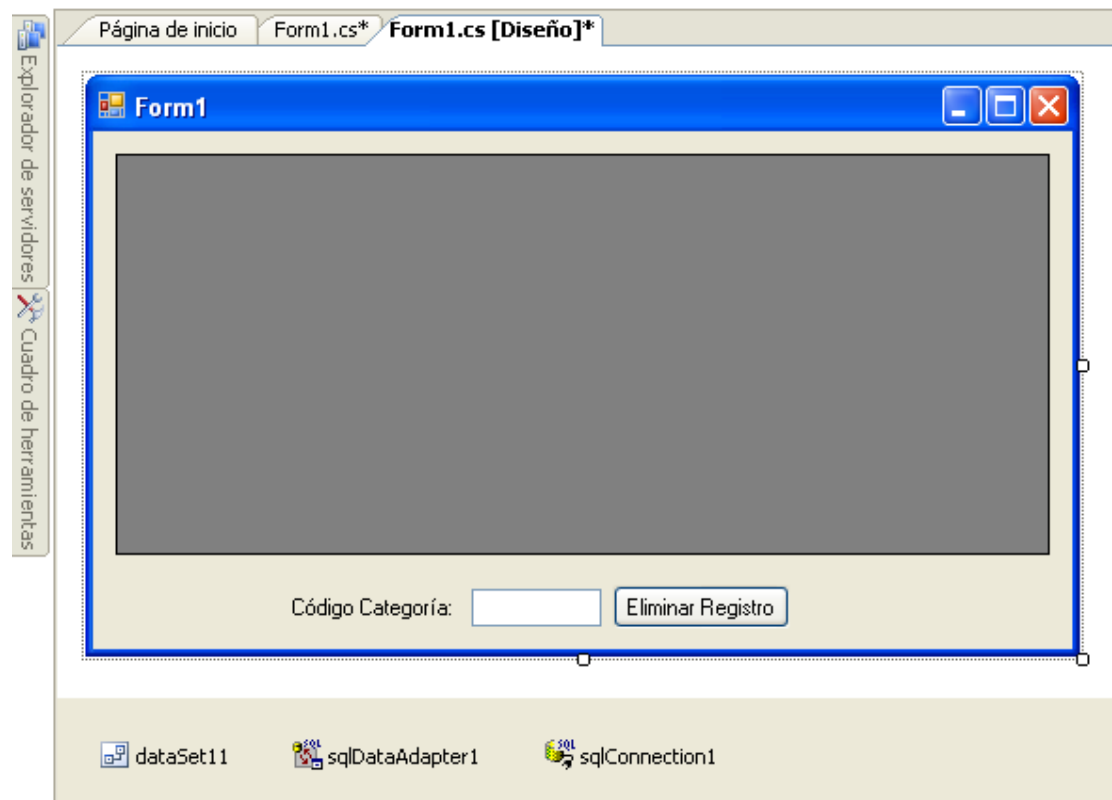
Ingrese Descripción de la Categoría:

Fila añadida correctamente

En la sección generada por Visual Studio en relación a DataSet1.Designer.cs se ha añadido un método sobrecargado para lograr la inserción sin considerar el grafico de la categoría. El código se detalla a continuación.

```
public CategoríasRow AddCategoríasRow(string NombreCategoría, string Descripción)
{
    CategoríasRow rowCategoríasRow = ((CategoríasRow)(this.NewRow()));
    rowCategoríasRow.ItemArray = new object[] {
        null,
        NombreCategoría,
        Descripción};
    this.Rows.Add(rowCategoríasRow);
    return rowCategoríasRow;
}
```

Diseñe otra interfaz y añada los controles necesarios para obtener la interfaz que se muestra en la siguiente figura ya que eliminaremos registros proporcionando el código de categoría.



```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace Sistema_Comercial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            sqlDataAdapter1.Fill(dataSet11, "Categorias");
            dataGridView1.DataSource = dataSet11.Categorias;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Eliminamos una de las filas
            dataSet11.Categorias.FindByIdCategoria(Convert.ToInt32(textBox1.Text)).Delete();
            //Actualizamos el dataSet
        }
    }
}

```

```

sqlDataAdapter1.Update(dataSet11);
MessageBox.Show("Fila eliminada correctamente");
    }
}

```

Nuestro ejemplo en ejecución es el que se muestra en la siguiente figura.



Ejemplo del uso de DataAdapter y DataSet tipado con todas las acciones de inserción y eliminación de datos incluidas

Como vemos, el uso del DataSet tipado, posee grandes ventajas sobre los DataSets no tipados, permitiéndonos trabajar con datos de forma muy rápida y sencilla. La parte más compleja quizás, es la preparación del esquema de datos, pero una vez realizada esta tarea, el trabajo con los datos es asombrosamente rápida y segura.

8.5.- ¿Qué son los datos Maestro Detalle?

El desarrollador de aplicaciones que debe trabajar con datos y fuentes de datos relacionadas entre sí, encuentra con frecuencia problemas de desarrollo en aplicaciones con datos interrelacionados.

Además, este tipo de aplicaciones, consumen gran parte del tiempo de desarrollo y son por lo general, acciones repetitivas.

Supongamos como ejemplo general, la tabla Categorías de un sistema comercial. Además, relacionemos las categorías del sistema comercial, con la tabla productos, para saber si un producto determinado está agrupado en una categoría, y en ese caso, saber cuáles.

Este sencillo ejemplo, es un claro exponente de una aplicación que relaciona datos maestro detalle.

Ambas tablas deben estar relacionadas para recopilar la información que se necesite en un momento dado.

Los datos maestros serían expuestos por las categorías del sistema comercial, mientras que los datos detalle estarían relacionados con los datos de los productos.

En nuestro caso, vamos a cambiar las palabras maestro y detalle por padre e hijo, y a partir de ahora, nos referiremos a padre como la tabla Categorías, e hijo como la tabla Productos. De esta forma, ubicaremos sin problemas ambos conceptos dentro del entorno de Visual Studio 2005, ya que éste tiene alguna ligera connotación que podría infundirnos a error como observará más adelante.

Por suerte, Visual Studio 2005 nos proporciona un conjunto de herramientas, que hace que realizar una aplicación Windows con todas las características de una aplicación maestro detalle, sea un auténtico juego de niños, que nos llevará aproximadamente un minuto de nuestro tiempo como mucho.

¿No me cree?. Continúe y se sorprenderá de lo que Visual Studio 2005 puede hacer por usted.

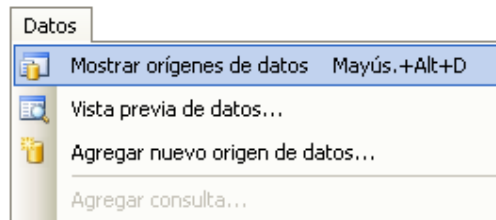
8.6.- Configurando la fuente de datos

Trabajar con fuentes de datos requiere como tarea inicial, que tengamos listo y preparado un origen de fuentes de datos válido.

Para esta tarea, deberemos configurar la fuente de datos que vamos a utilizar, algo que vamos a aprender a hacer a continuación.

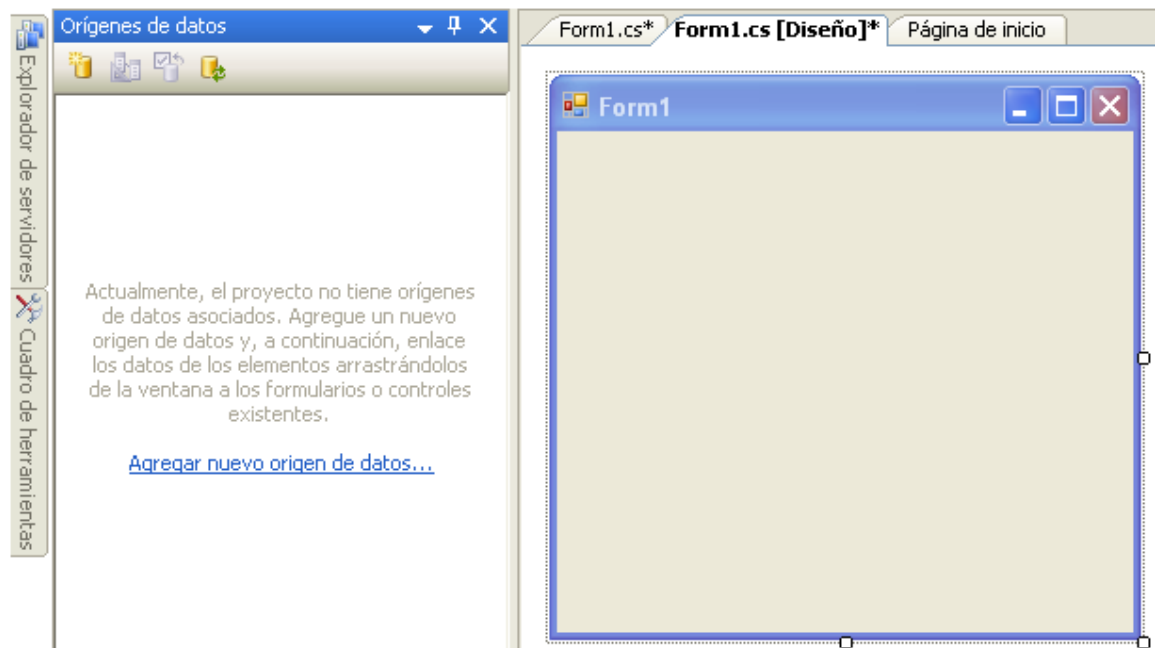
Configurando el origen de la fuente de datos

Iniciaremos una nueva aplicación Windows con Visual Studio 2005 y seleccionaremos el menú Data > Show Data Sources (Datos > Mostrar orígenes de datos) como se indica en la figura.



Menú para mostrar los orígenes de datos

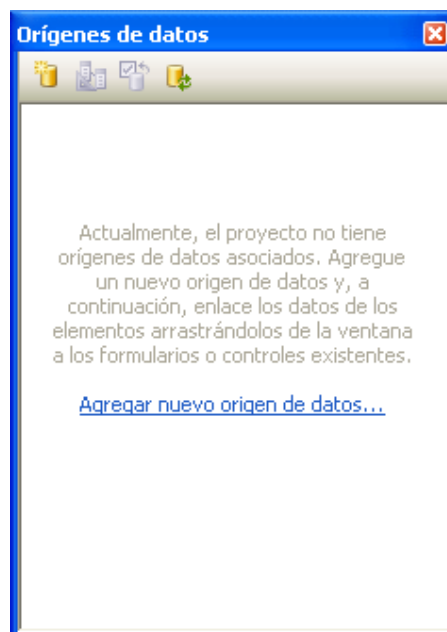
En este punto, aparecerá una ventana como la que se muestra en la siguiente figura.



Ventana de orígenes de datos

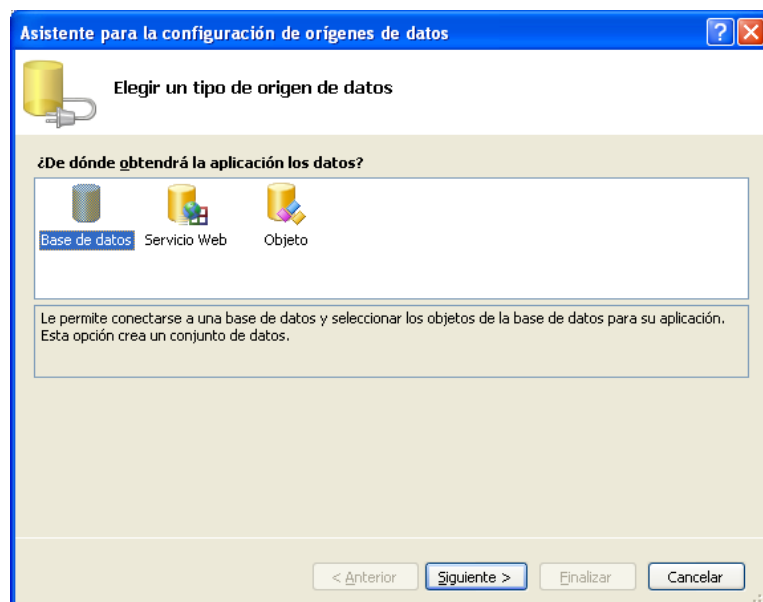
Como podemos apreciar, la ventana no tiene por defecto ningún origen de datos asignado, además, de no estar anclada a ningún sitio del entorno, deberá anclarla.

Sobre la configuración del origen de datos, haga clic sobre la opción Agregar nuevo origen de datos... como se indica en la figura.



Ventana de orígenes de datos

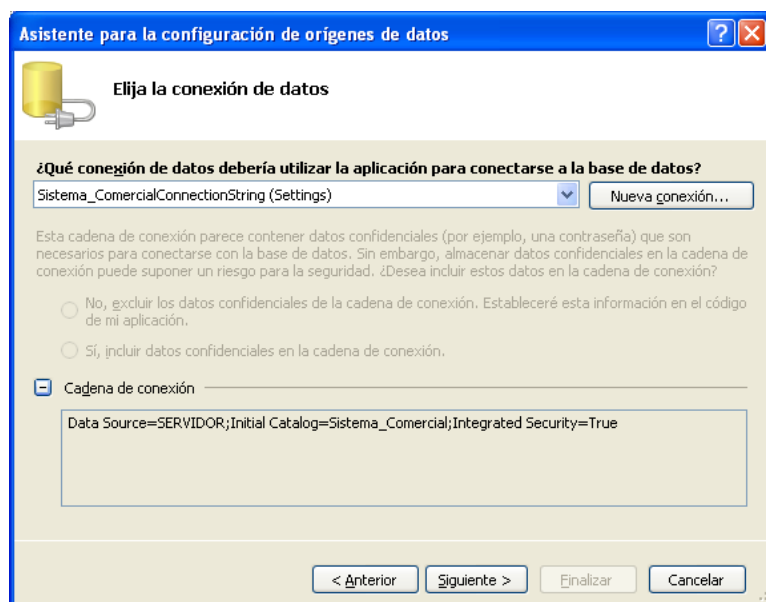
Aparecerá una ventana como la que se muestra en la siguiente figura en la cuál indicaremos el lugar de dónde la aplicación obtendrá los datos, y que en nuestro caso será de una Base de datos.



Como tipo de origen de datos elegiremos una Base de datos

Una vez seleccionado la opción de Base de datos como tipo de origen de datos, presionaremos el botón Siguiente.

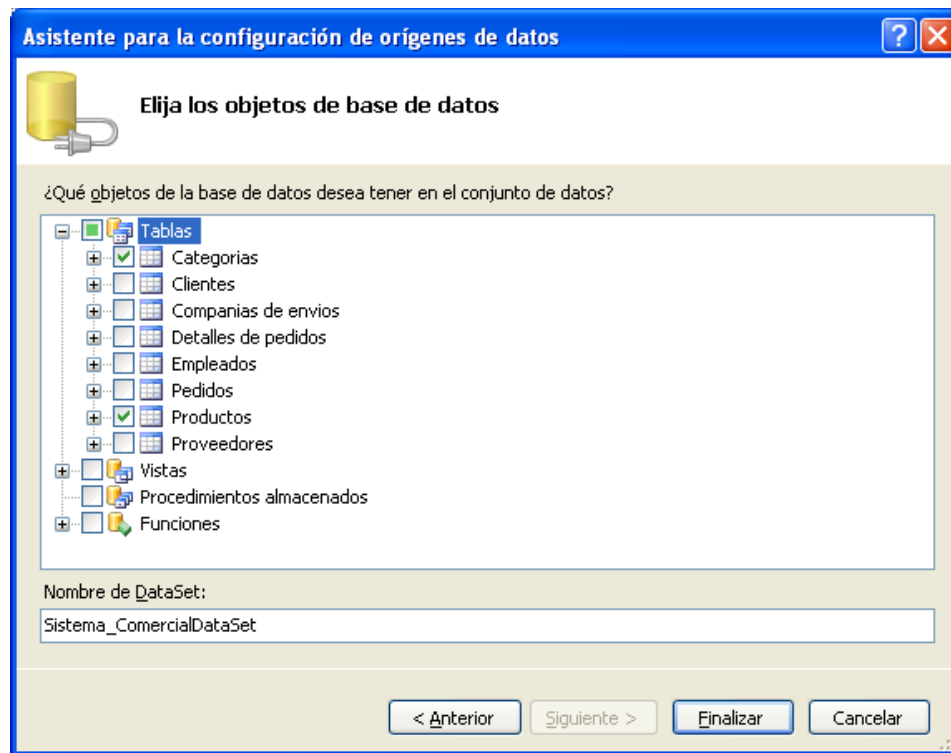
En la siguiente ventana, elegiremos la conexión de la base de datos que vamos a utilizar, o presionaremos sobre el botón Nueva conexión... sino tenemos seleccionada la conexión que queremos utilizar. A continuación, podemos ver como hemos establecido la conexión con nuestra fuente de datos, que utilizaremos en nuestro ejemplo de creación de la aplicación de acceso a datos maestro detalle.



Ventana dónde elegimos la conexión de datos

Seguidamente, haremos clic en el botón Siguiente.

En la nueva ventana que aparece ahora dentro del asistente para la creación del origen de datos. A continuación, haremos clic sobre el botón Siguiente.



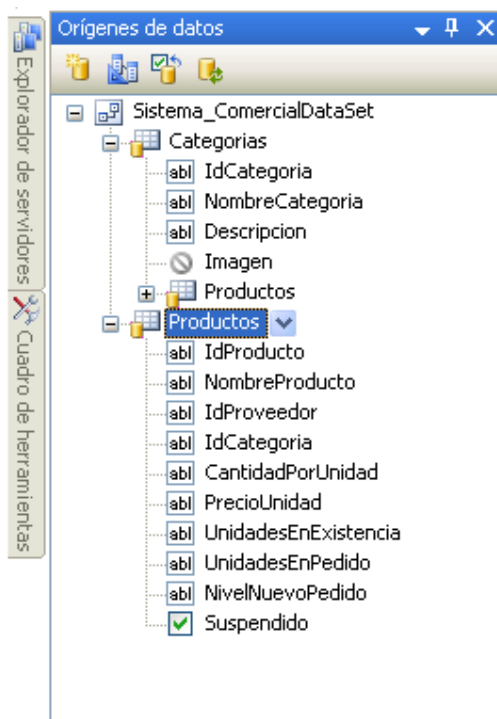
Ventana con los objetos de la base de datos seleccionada

En este punto, el asistente se conecta a la base de datos para recuperar la información de la base de datos y mostrarla en la ventana del asistente como se muestra en la figura anterior.

A continuación, despliegue el objeto Tablas y seleccione las tablas Categorías y Productos como se indica en la figura anterior.

Una vez que hemos seleccionado los objetos de la base de datos, haremos clic sobre el botón Finalizar para que el asistente concluya las opciones añadidas al asistente.

La ventana del origen de datos, quedará ahora configurado de una forma similar a la que se presenta en la siguiente figura.



Ventana de origen de datos con la configuración añadida

A partir de aquí, deberemos preparar las tablas del origen de datos para que se comporten como auténticos datos e informaciones maestro detalle.

8.7.- Preparando el origen de datos

Ya hemos aprendido a añadir nuestro origen de datos, y ahora aprenderemos a prepararlo para poder utilizarlo posteriormente en la aplicación Windows.

La preparación del origen de datos, nos permitirá seleccionar que tabla o datos queremos que actúen como maestro y cuales como detalle, o dicho de otra forma, que datos queremos que sean padre y cuales hijo.

Nuestro objetivo principal es mostrar la tabla Productos como tabla hijo y la tabla Categorías como padre de la tabla anterior.

Prepararemos e incrustaremos primero la tabla Categorías dentro del formulario Windows como Detalle de la información, y posteriormente insertaremos la tabla Productos dentro del formulario.

Por último, asignaremos alguna relación que permita trabajar con las dos tablas en nuestro formulario Windows sin perder la conexión entre ambas tablas y permitiéndonos acceder a la información que nos proporciona dicha relación.

Preparando la tabla padre

Lo primero que haremos será preparar la tabla padre para poderla añadir al formulario Windows.

Por esa razón, haremos clic sobre la tabla Categorías de la ventana de Orígenes de datos como se indica en la figura.

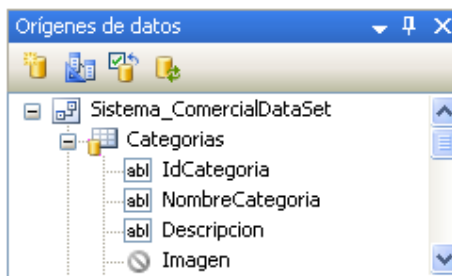
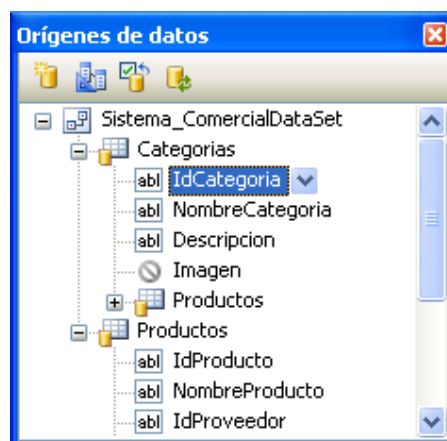


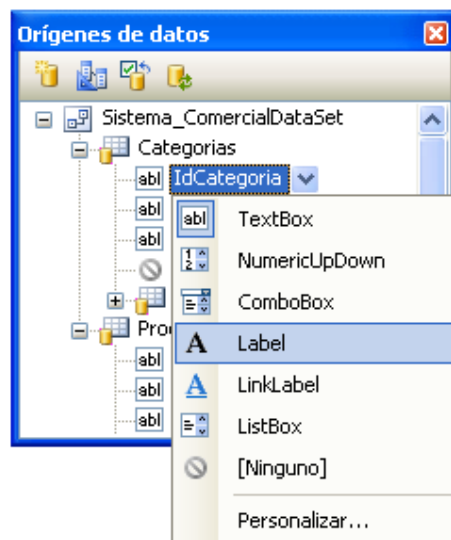
Tabla Categorías de la ventana de orígenes de datos

En la ventana de Orígenes de datos y en concreto con la tabla Categorías desplegada, centraremos nuestra atención en el campo IdCategoria como se indica en la figura.



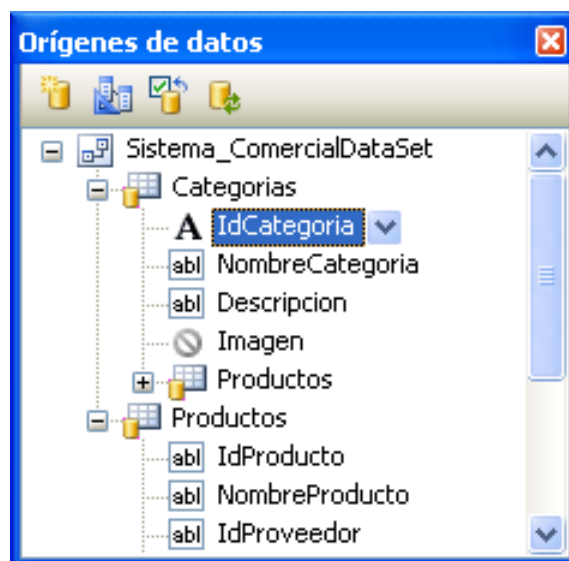
Campo IdCategoria de la tabla Categorías

Pulse sobre la lista desplegable que aparece a la derecha del campo IdCategoria y seleccione la opción Label como se indica en la figura.



Lista desplegable con la opción Label seleccionada como campo de la tabla desplegada

En este caso, la ventana de Orígenes de datos quedará informada tal y como se indica en la siguiente figura.



Campo IdCategoria modificado en la ventana de Orígenes de datos

A continuación, haremos clic sobre la tabla Categorias como se indica en la figura, y posteriormente presionaremos sobre la lista desplegable que aparece a la derecha de la tabla.

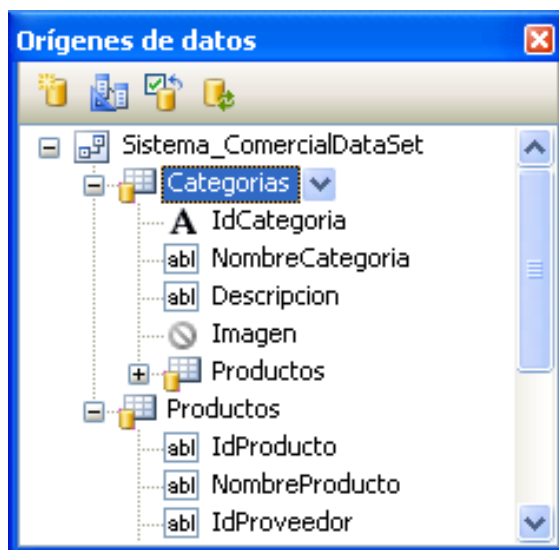
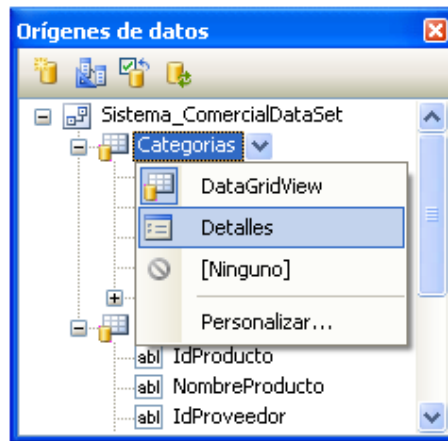


Tabla Categorías seleccionada en la ventana de Orígenes de datos

Si hacemos clic sobre la lista desplegable, aparecerá una lista de opciones o posibilidades, para indicar cómo queremos que sean los campos de la tabla seleccionada con respecto a que tipo de controles queremos que sean.

Esto es lo que se indica en la siguiente figura.



Lista desplegable de la tabla Categorías de la ventana de Orígenes de datos

Por defecto, una tabla queda determinada con un icono que representa el control DataGridView, aunque se puede modificar la representación que deseamos tengan los datos dentro de un formulario seleccionando cualquiera de las opciones que tenemos de la lista desplegable.

Estas opciones pueden ser cualquiera de las siguientes:



Representa los datos volcados dentro de un control DataGridView



Representa los datos volcados dentro de controles estándar como TextBox u otros controles para reflejarla como Detalle de la información



No representa ningún control como tipo de control de volcado de datos

En el caso de la tabla Categorías que usaremos como tabla padre, cambiaremos la representación por defecto de DataGridView para indicarle que nos represente la información en controles, tal y como se indica en la figura.

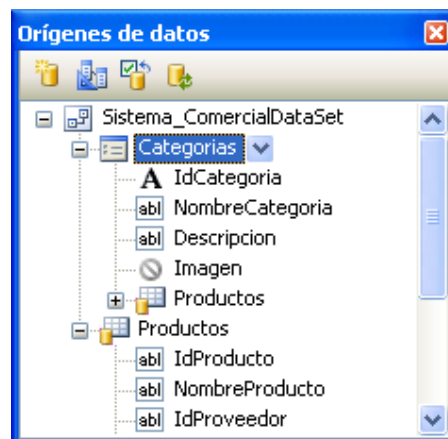


Tabla Categorías seleccionada en la ventana de Orígenes de datos

Ahora que tenemos la tabla maestra ya preparada, pasaremos a preparar la tabla hija.

Preparando la tabla hija

Ahora que ya tenemos preparada la tabla padre, prepararemos la tabla hija de los productos del sistema comercial, para poder usar su información dentro del formulario Windows.

Por esa razón, haga clic sobre la tabla Productos de la ventana de Orígenes de datos como se indica en la figura.

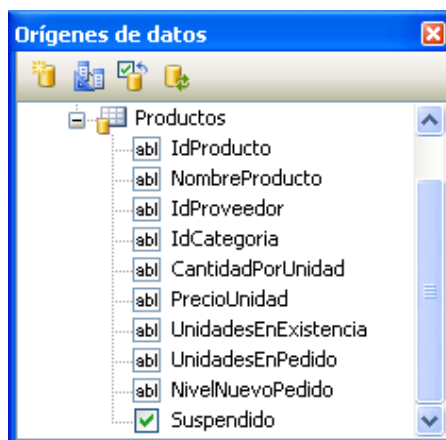
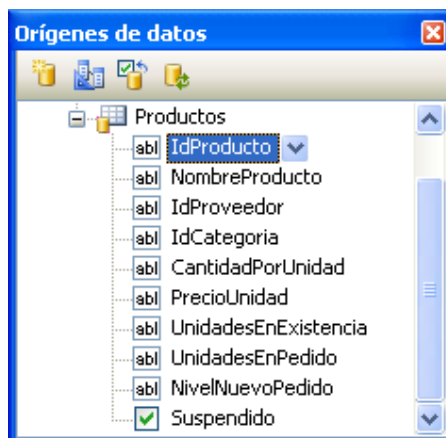


Tabla Productos de la ventana de orígenes de datos

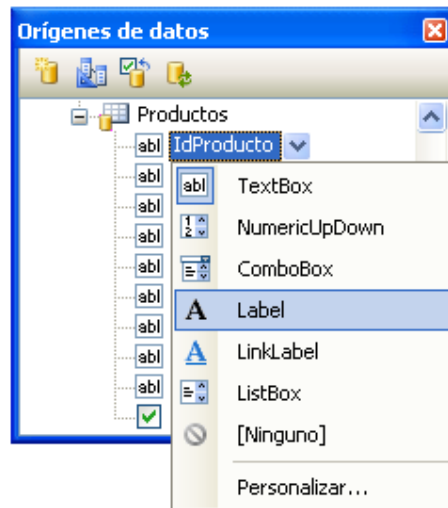
Dentro de la ventana de Orígenes de datos y en concreto de la tabla Productos desplegada, centraremos nuestra atención en el campo IdProducto como se indica en la figura.



Campos IdProducto de la tabla Productos

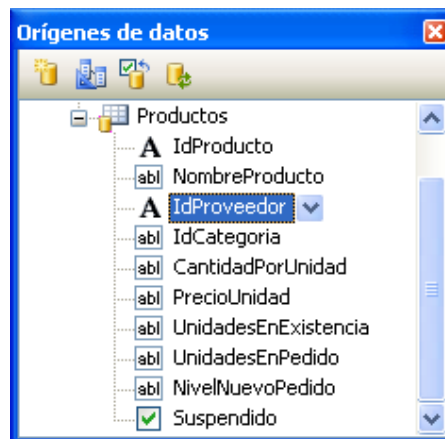
Sobre el campo IdProducto, haremos una pequeña modificación.

Pulse sobre la lista desplegable que aparece a la derecha del campo IdProducto y seleccione la opción Label como se indica en la figura.



Lista desplegable de opciones de un campo de la tabla desplegada

Del mismo modo para el campo IdProveedor. De esta manera, el campo IdProducto e IdProveedor quedará modificado en la ventana Orígenes de datos como se indica en la figura.



Campo IdProducto e IdProveedor modificados en la ventana de Orígenes de datos

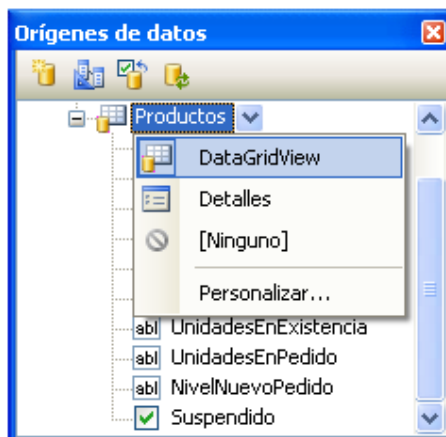
A continuación, haremos clic sobre la tabla Productos como se indica en la siguiente figura, y posteriormente presionaremos sobre la lista desplegable que aparece a la derecha de la tabla.



Tabla Productos seleccionada en la ventana de Orígenes de datos

Nos aseguraremos que está seleccionada la opción DataGridView que es la que aparece por defecto.

Esto es lo que se indica en la figura.



En la tabla Productos, nos aseguraremos de seleccionar la opción DataGridView

Una vez que tenemos las tabla maestra y detalle preparadas para utilizarlas, las añadiremos al formulario Windows para que tengan el funcionamiento esperado.

8.8.- Incrustando los datos maestro detalle

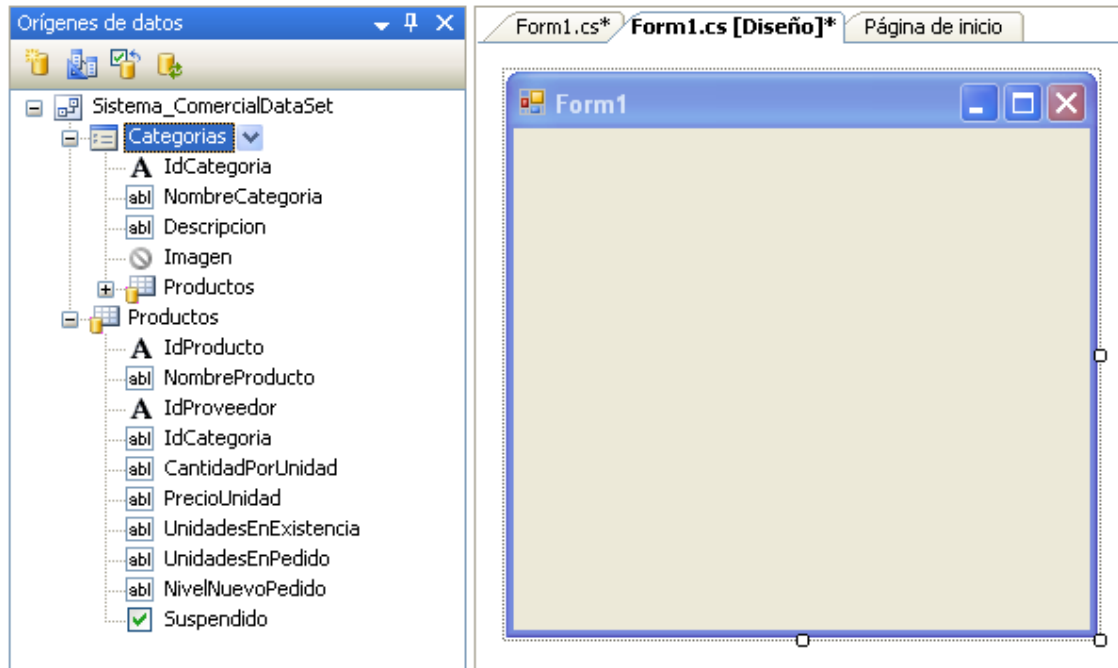
Ya sabemos como crear un origen de datos, también sabemos como preparar los datos maestro y detalle, y por último, lo que nos queda es insertar esos datos en el formulario, y relacionar todos sus datos para que funcionen de la forma esperada.

A continuación, veremos como llevar a cabo esta tarea y aprenderemos a hacerlo posible de forma muy rápida y sencilla.

Incrustando la tabla padre en el formulario

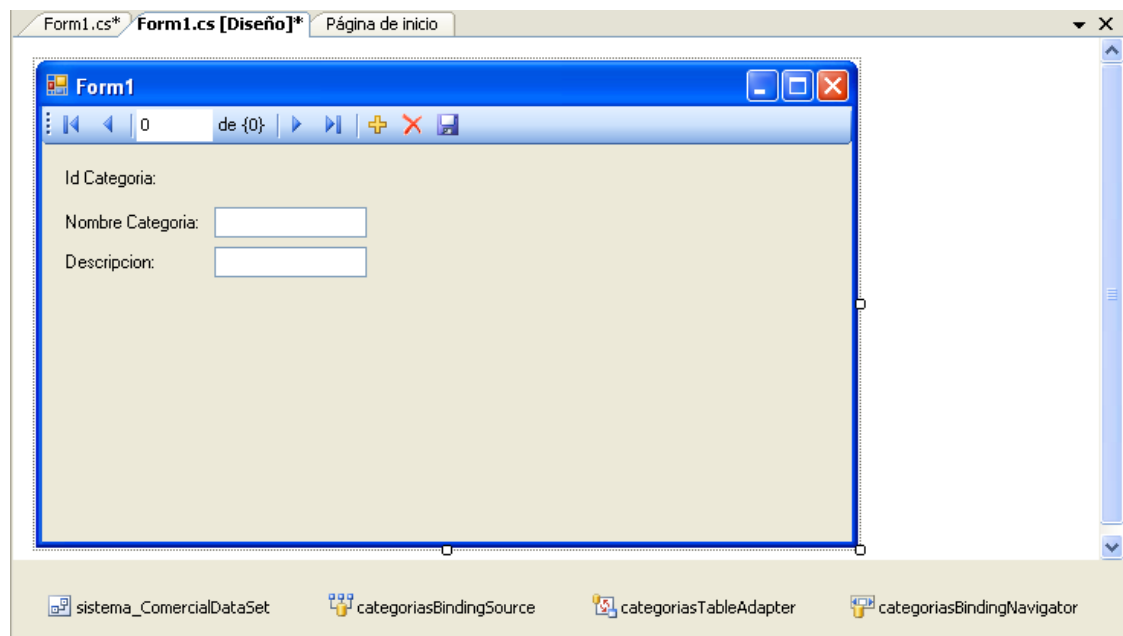
Nuestra primera acción, será incrustar en el formulario los datos o tabla padre, que en nuestro caso es la formada por la tabla Categorías.

Para situar los datos de la tabla Categorías dentro de un formulario y en concreto como una información de maestro, bastará con arrastrar y soltar la tabla Categorías sobre el formulario como se indica en la figura.



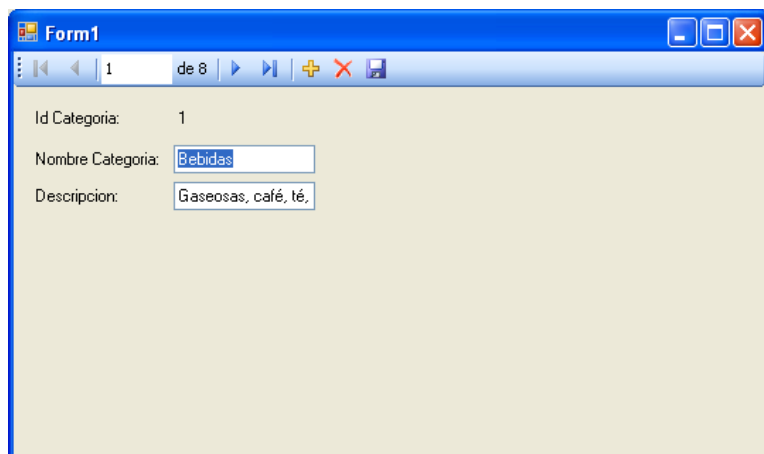
Para presentar la información como detalle, arrastraremos la tabla Categorías de la ventana Orígenes de datos sobre el formulario Windows

Observaremos que Visual Studio 2005, generará por nosotros un conjunto de componentes y controles, que por defecto tendrá una apariencia similar a la que se presenta en la siguiente figura.



Controles y Componentes de la tabla maestra añadidos al formulario Windows

Si ejecutamos nuestra aplicación, observaremos que esta actúa como una típica aplicación de acceso a datos que nos permite navegar a través de los campos de la tabla Categorías, tal y como se indica en la siguiente figura.



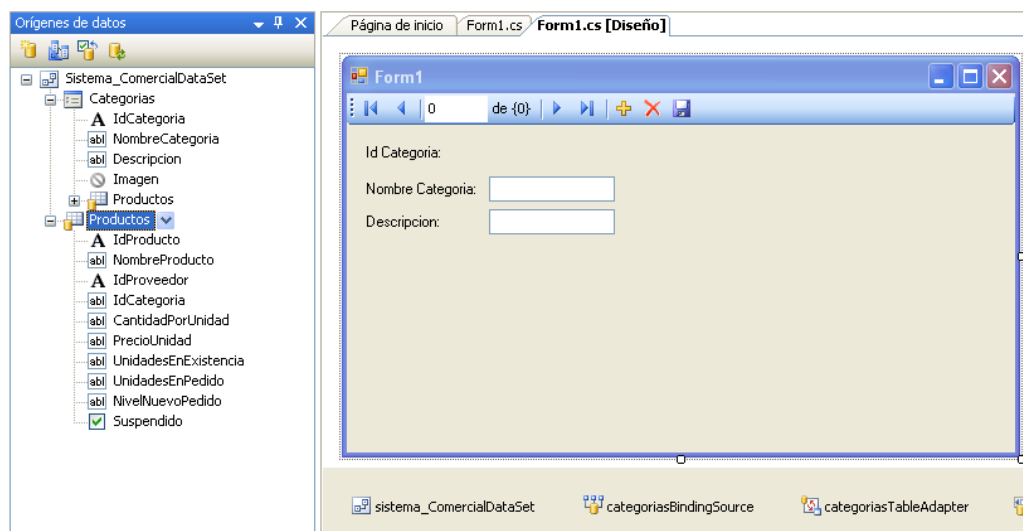
Aplicación en ejecución de la tabla de detalle incrustada en el formulario Windows

A continuación, insertaremos en el formulario la tabla Productos y relacionaremos ambas tablas para que se muestren los datos relacionados, dentro del formulario Windows.

Incrustando la tabla hija en el formulario

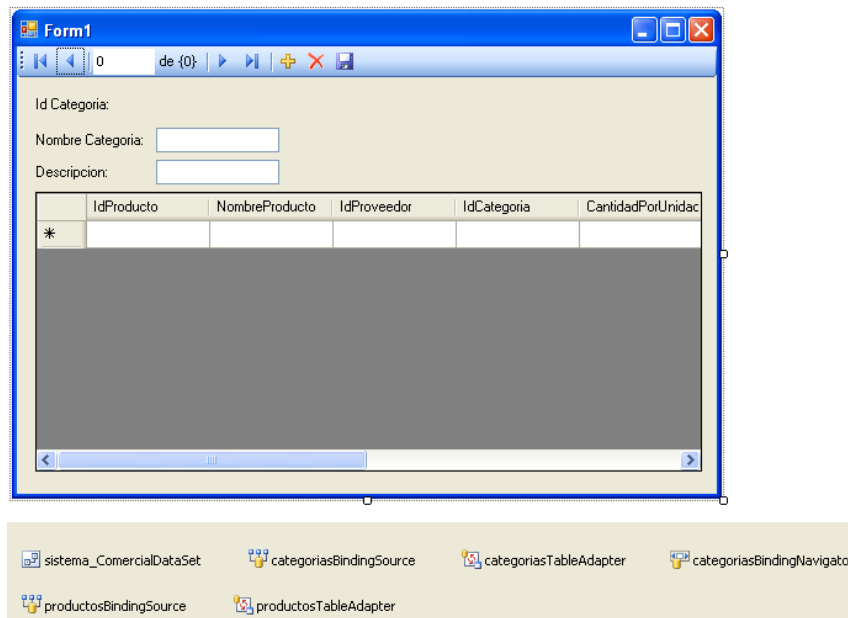
Ya tenemos la tabla padre insertada en nuestro formulario Windows. Nuestra segunda acción, será la de incrustar en el formulario los datos o tabla hija, que en nuestro caso es la formada por la tabla Productos, la cuál además, posee una relación entre campos con la tabla Categorías insertada anteriormente.

Para llevar a cabo esta acción arrastraremos y soltaremos la tabla Productos sobre el formulario como se indica en la figura.



Para presentar la información de la tabla Productos, arrastraremos la tabla de la ventana Orígenes de datos sobre el formulario Windows

Observe que Visual Studio 2005, genera por nosotros más componentes y controles, que por defecto tendrá una apariencia similar a la que se presenta a continuación.



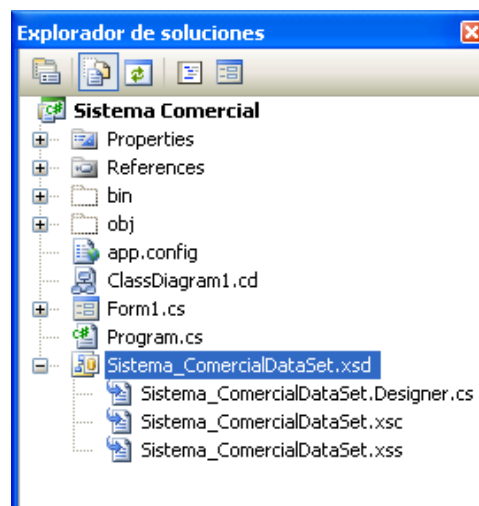
Controles y Componentes de la tabla maestra y detalle añadidos al formulario Windows

Como podemos observar, el entorno de trabajo ha hecho por nosotros el trabajo más complejo para representar los datos de forma rápida y sencilla.

El esquema de datos tipados, aparecía ya en nuestro proyecto cuando asignamos el correspondiente origen de datos.

Ahora lo que ha ocurrido, es que al arrastrar y soltar la tabla padre Categorías de la ventana de Orígenes de datos, en el entorno se ha añadido un componente de nombre `sistema_comercialDataSet` que es el que permitirá relacionar el DataSet tipado con nuestros datos.

Este componente será usado por la relación maestro detalle de las dos tablas añadidas al formulario. En la siguiente figura, podemos ver el esquema añadido a nuestro proyecto, y el componente del que estamos hablando.



Esquema del DataSet tipado añadido al proyecto y su componente de relación

Ejecute la aplicación y observe el comportamiento de la misma.

Observará por lo tanto, que los datos entre detalle y maestra, no están relacionados.

Si navegamos a través de los datos de detalle a través del objeto categoriasBindingNavigator, el control DataGridView no representa la relación de los datos seleccionados.

Esto es lo que se muestra en la siguiente figura.

	IdProducto	NombreProducto	IdProveedor	IdCategoria	CantidadPorUnid
▶	1	Té Dharamsala	1	1	10 cajas x 20 bo
	2	Cerveza tibetana ...	1	1	24 - bot. 12 l
	3	Sirope de regaliz	1	2	12 - bot. 550 ml
	4	Espicias Cajun d...	2	2	48 - frascos 6 l
	5	Mezcla Gumbo d...	2	2	36 cajas
	6	Mermelada de gr...	3	2	12 - frascos 8 l
	7	Peras secas orgá...	3	7	12 - paq. 1 kg
	8	Salsa de arándan...	3	2	12 - frascos 12 l

Ejecución de la aplicación confirmando que los datos mostrados no están relacionados

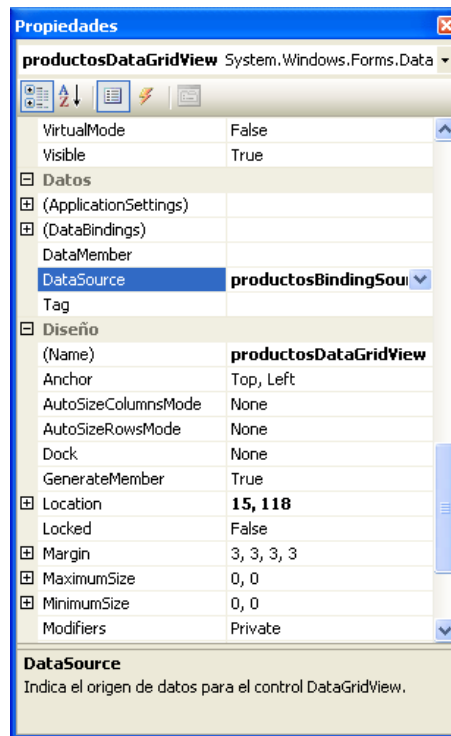
A continuación, la tarea que nos queda para completar el correcto funcionamiento de nuestra aplicación, es la de relacionar la tabla detalle y la tabla maestra entre sí, para que los datos que se muestran en la aplicación, estén relacionados entre sí.

Relacionando la tabla padre con la tabla hija

La tarea más sencilla es la de relacionar la tabla detalle con la tabla maestra. Es una tarea sencilla, porque Visual Studio 2005 nos proporciona las herramientas necesarias para simplificar al máximo esta tarea.

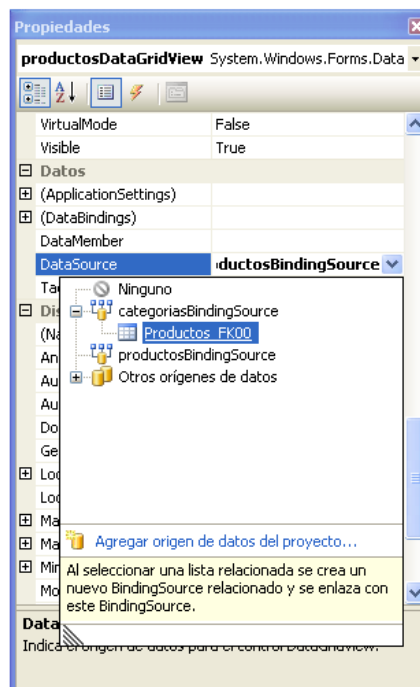
Para llevar a cabo esta tarea, haga clic sobre el control DataGridView que corresponde a los datos de la tabla hija, y acceda a la ventana de Propiedades.

Dentro de la ventana de Propiedades, acceda a la propiedad DataSource como se indica en la figura.



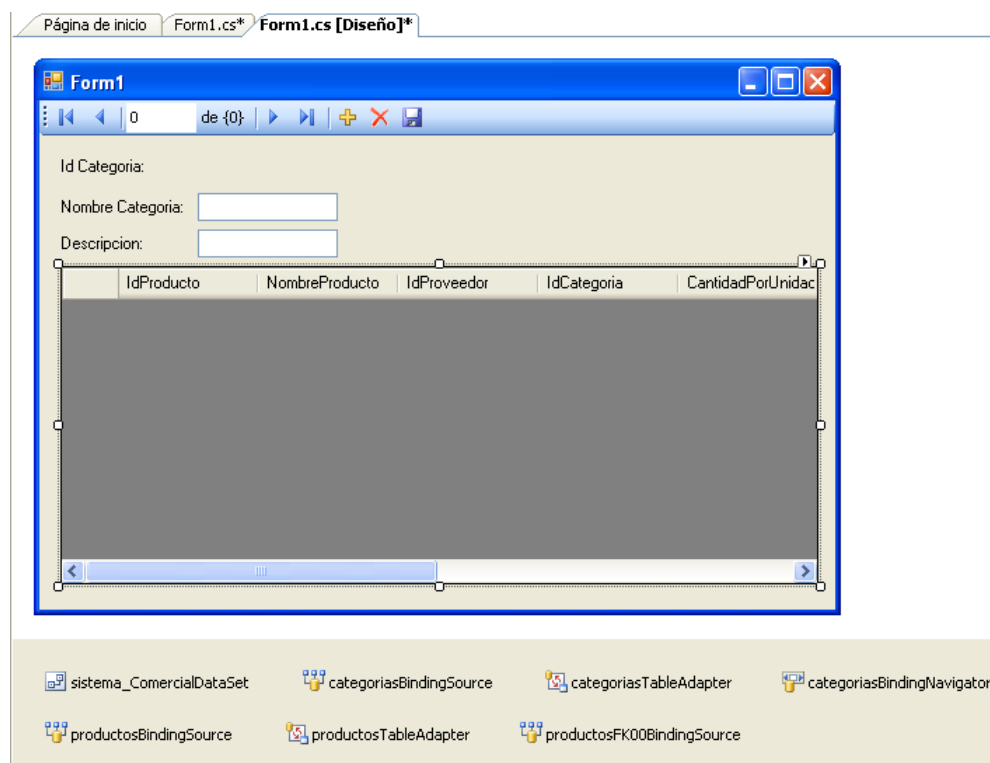
Propiedad DataSource del control DataGridView de la información maestra

Despliegue esta propiedad, y de la lista desplegable que aparece, seleccione la opción ProductosFK00 como se indica en la figura.



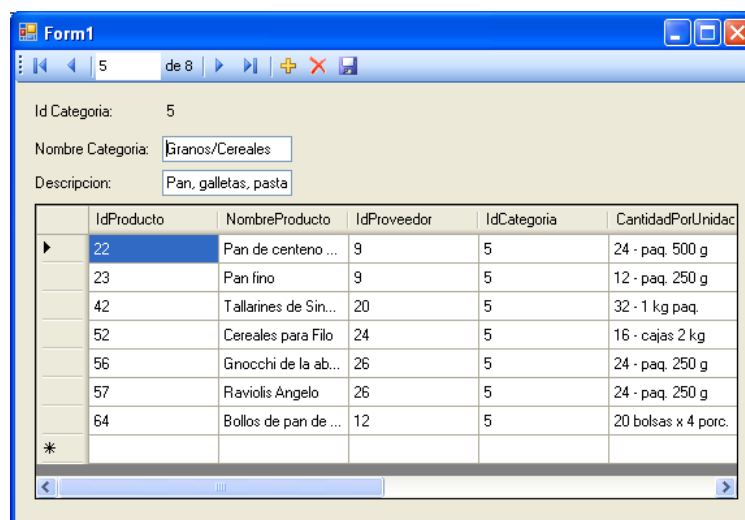
Asignación de la clave de relación entre las tablas

Cuando se asigna el campo de relación de las tablas, dentro de la aplicación se añade esta relación para que cuando naveguemos entre los datos de la tabla Categorías aparezca toda la información de la tabla Productos relacionada con la tabla Categorías. Esto de lo que hablamos, está supeditado por el componente productosFK00BindingSource que es lo que se indica en la siguiente figura.



Controles y componentes incluido el de relación entre tablas, añadidos al formulario Windows

Para finalizar, ejecutaremos nuestra aplicación y comprobaremos que el funcionamiento de esta, incluida la relación entre tablas, funciona como esperábamos. En la figura que se muestra a continuación, podemos observar el comportamiento de nuestra aplicación en ejecución.



Aplicación en ejecución, mostrando la correcta relación entre las tablas

8.9.- Manipulando los datos maestro detalle

Obviamente, los datos maestro detalle no nos sirve únicamente para insertar las tablas de datos en un formulario, mostrarlos y navegar por ellos.

Además de esto, podemos también manipular los datos maestro detalle, modificarlos, actualizarlos, borrarlos, sin hacer ninguna acción adicional.

El control BindingNavigator ya proporciona todas las características necesarias para realizar estas acciones.

Podemos personalizar el control para permitir o denegar estas acciones.

Además, dentro de la ventana de Orígenes de datos, podemos seleccionar diferentes campos de las tablas y cambiar el tipo de control en el que queremos representar sus datos.

A continuación veremos un breve ejemplo de como manipular datos para que nos sirva de aprendizaje de cómo hacer esto posible.

Modificando datos

Ejecute la aplicación de ejemplo que hemos diseñado hasta ahora y sitúese en alguno de sus campos. Centrándonos en la información de la tabla Categorías, cambiaremos un campo determinado, como el que se muestra en la figura.

Acto seguido, cuando hayamos realizado la modificación, haremos clic sobre la opción de Guardar datos, tal y como se muestra en la figura.

	IdProducto	NombreProducto	IdProveedor
	22	Pan de centeno ...	9
▶	23	Pan fino	9

Opción del control BindingNavigator para guardar los datos modificados

Como vemos, la manipulación de datos es realmente sencilla y en la relación de datos mostrada, no tiene porqué presentarnos ninguna dificultad.

Insertando y eliminando datos

Si queremos agregar datos, deberemos hacer clic sobre la opción Agregar nuevo del control BindingNavigator como se muestra en la siguiente figura.

Form1

6 de 8

Id Categoría: 6

Nombre Categoría: Carnes

Descripción: Carnes preparadas

Agregar nuevo

	IdProducto	NombreProducto	IdProvee
▶	9	Buey Mishi Kobe	4
	17	Cordero Alice Spr...	7
	29	Salchicha Thürin...	12

Añadir un registro nuevo es realmente sencillo

De la misma manera funciona el mecanismo para eliminar un registro, tal y como se muestra en la siguiente figura.

Form1

6 de 8

Id Categoría: 6


Nombre Categoría: Carnes

Descripción: Carnes preparadas

Eliminar

	IdProducto	NombreProducto	IdPr
▶	9	Buey Mishi Kobe	4
	17	Cordero Alice Spr...	7
	29	Salchicha Thürin...	12

Eliminar un registro de forma rápida

Recuerde presionar el icono  si quiere que los cambios y modificaciones realizadas se mantengan. Pulsando sobre ese icono, la acción de manipulación de datos se lanza contra la base de datos.



RESUMEN

El desarrollador de sistema de información que trabaja con datos y fuentes de datos relacionadas entre sí, encuentra con frecuencia problemas de desarrollo en aplicaciones con datos interrelacionados.

Visual Studio proporciona a los desarrolladores un conjunto de herramientas y asistentes que facilitan enormemente el trabajo y ahorran grandes cantidades de tiempo. De los asistentes o herramientas que nos proporcionan mayor rendimiento en el entorno de desarrollo rápido de Microsoft, está la que nos permite trabajar con fuentes de datos como los datos de una aplicación típica maestro detalle.

**BIBLIOGRAFÍA RECOMENDADA**

Visual C# 2005 – *Charte Francisco* – Editorial ANAYA MULTIMEDIA-ANAYA INTERACTIVA.

Enciclopedia de Microsoft Visual C# – *Ceballos, FJ* – Editorial RAMA.

C# Desarrollo con Visual Studio 2005 – *Hervé Berthet* – Editorial Recursos Informáticos.

Microsoft Visual C++ 6.0 Manual de Programador – *Beck Zaratian* – Microsoft Press.

SQL Server 2005 – *Jérôme Gabillaud* – Editorial Recursos Informáticos.

Aplicando SQL Server 2000 – *Rafael Juan Cherre* – Editorial Macro.

**NEXO**

Esta unidad TEMÁTICA proporciona información detallada para la implementación de aplicaciones con datos interrelacionados.

**ACTIVIDAD**

En la organización que está desarrollando el sistema de información implemente formularios que utilicen el esquema maestro detalle para mostrar información de las diferentes tablas que lo requieran.

Poner en práctica todos los programas descritos en la unidad TEMÁTICA.

**AUTOEVALUACIÓN FORMATIVA**

Desarrolla las siguientes aplicaciones utilizando Microsoft Visual C#:

- 1.- Diseñe una aplicación Windows que permita gestionar el expendio de productos para la base de datos del sistema comercial. Debe considerar datos de Clientes, búsqueda de productos, cálculos de impuestos e importe a pagar.