## SEMANA 3

# ACCESO A DATOS CON ADO.NET 2.0

**¿Qué es ADO.NET?**

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos de aplicaciones para usuario y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

Las clases de ADO.NET se encuentran en el archivo System.Data.dll y están integradas con las clases de XML que se encuentran en el archivo System.Xml.dll. Cuando se compila un código que utiliza el espacio de nombres System.Data, es necesario hacer referencia a los archivos System.Data.dll y System.Xml.dll. Para obtener un ejemplo de una aplicación de ADO.NET que se conecta a una base de datos, recupera datos de ésta y, a continuación, los muestra en el símbolo del sistema.

ADO.NET proporciona funcionalidad a los programadores que escriben código administrado similar a la funcionalidad que los objetos ADO (ActiveX Data Objects) proporcionan a los programadores de modelo de objetos componentes (COM) nativo.

### Objetivos de diseño para ADO.NET

A medida que la programación de aplicaciones ha evolucionado, las nuevas aplicaciones se han convertido en aplicaciones de correspondencia imprecisa basadas en el modelo de aplicación Web. Las aplicaciones de hoy en día utilizan cada vez más XML para codificar datos que se van a pasar a través de conexiones de red. Las aplicaciones Web utilizan HTTP para las comunicaciones entre niveles y, por tanto, deben controlar expresamente el mantenimiento del estado de una solicitud a otra. Este nuevo modelo es muy diferente del estilo de programación con conexión y de correspondencia precisa que caracterizaba la época cliente-servidor, en la que una conexión permanecía abierta durante toda la vida del programa y no hacía falta controlar el estado.

A la hora de diseñar herramientas y tecnologías para satisfacer las necesidades del programador de hoy en día, Microsoft se dio cuenta de que hacía falta un modelo de programación totalmente nuevo para el acceso a datos, un modelo basado en .NET Framework. Tomar .NET Framework como base garantizaba que la tecnología de acceso a datos sería uniforme: los componentes compartirían un sistema de tipos, unos modelos de diseño y unas convenciones de nomenclatura.

ADO.NET se diseñó para cumplir con los objetivos de este nuevo modelo de programación: arquitectura de datos sin mantener una conexión abierta, estrecha integración con XML, representación común de datos con la posibilidad de combinar datos procedentes de múltiples y variados orígenes, y servicios optimizados para interactuar con una base de datos, todo ello nativo de .NET Framework.

A la hora de crear ADO.NET, Microsoft se propuso los siguientes objetivos de diseño:

- Aprovechar la tecnología de objetos ADO (ActiveX Data Objects) actuales.
- Admitir el modelo de programación n-tier
- Integrar la compatibilidad con XML

### Aprovechar los conocimientos actuales de ADO

El diseño de ADO.NET satisface muchos de los requisitos del modelo de desarrollo de aplicaciones de hoy en día. Al mismo tiempo, el modelo de programación permanece similar a ADO, en la medida de lo posible, de manera que los actuales programadores de ADO no tengan que comenzar desde el principio. ADO.NET forma parte intrínseca de .NET Framework y al programador de ADO le sigue resultando familiar.

ADO.NET también coexiste con ADO. Aunque la mayoría de las nuevas aplicaciones basadas en .NET se escribirán mediante ADO.NET, ADO sigue estando disponible para el programador de .NET a través de los servicios de interoperabilidad COM de .NET.

### Admitir el modelo de programación N-Tier

La idea de trabajar con un conjunto de datos sin mantener una conexión abierta se ha convertido en un objetivo del modelo de programación. ADO.NET proporciona compatibilidad de primera clase con el entorno de programación n-tier sin mantener una conexión abierta para el que están escritas muchas aplicaciones nuevas. La solución de ADO.NET para la programación n-tier es el DataSet.

### Integrar la compatibilidad con XML

XML y el acceso a datos están estrechamente relacionados. XML trata la codificación de datos y el acceso a datos trata cada vez más sobre XML. .NET Framework no sólo admite los estándares Web, sino que está basado totalmente en ellos.

La compatibilidad con XML está integrada en los cimientos de ADO.NET. Las clases de XML incluidas en .NET Framework y ADO.NET forman parte de la misma arquitectura: están integradas en muchos niveles. Ya no es necesario elegir entre el conjunto de servicios de acceso a datos y los correspondientes servicios de XML; la capacidad para cruzar de uno a otro es inherente al diseño de ambos.

### Arquitectura de ADO.NET

Tradicionalmente, el procesamiento de datos ha dependido principalmente de un modelo de dos niveles basado en una conexión. A medida que el procesamiento de datos utiliza cada vez más arquitecturas de varios niveles, los programadores están pasando a un enfoque sin conexión con el fin de proporcionar una escalabilidad mejor para sus aplicaciones.

### Componentes de ADO.NET

Existen dos componentes de ADO.NET que se pueden utilizar para obtener acceso a datos y manipularlos:

- Proveedores de datos de .NET Framework
- El DataSet

#### Proveedores de datos de .NET Framework

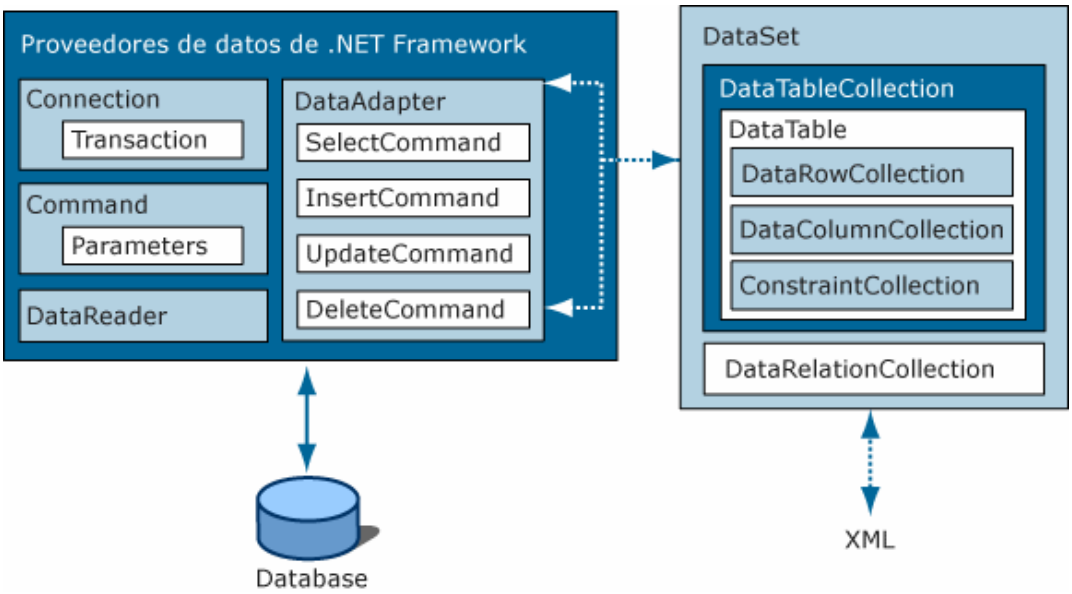
Los proveedores de datos de .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de sólo lectura y sólo avance. El objeto **Connection** proporciona conectividad a un origen de datos. El objeto **Command** permite tener acceso a comandos de base de datos para devolver datos, modificar datos, ejecutar procedimientos almacenados y enviar o recuperar información sobre parámetros. El objeto **DataReader** proporciona una secuencia de datos de alto rendimiento desde el origen de datos. Por último, el objeto DataAdapter proporciona el puente entre el objeto **DataSet** y el origen de datos. El **DataAdapter** utiliza objetos **Command** para ejecutar comandos SQL en el origen de datos tanto para cargar el **DataSet** con datos como para reconciliar en el origen de datos los cambios aplicados a los datos incluidos en el **DataSet**.

#### DataSet

El DataSet de ADO.NET está expresamente diseñado para el acceso a datos independientemente del origen de datos. Como resultado, se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. El **DataSet** contiene una colección de uno o más objetos DataTable formados por filas y columnas de datos, así como información sobre claves principales, claves externas, restricciones y relaciones relativa a los datos incluidos en los objetos **DataTable**.

En el diagrama siguiente se ilustra la relación entre un proveedor de datos de .NET Framework y un **DataSet**.

### Arquitectura de ADO.NET



### Elegir un DataReader o un DataSet

A la hora de decidir si su aplicación debe utilizar un **DataReader** (vea Recuperar datos mediante DataReader) o un **DataSet** (vea Utilizar DataSets en ADO.NET), debe tener en cuenta el tipo de funcionalidad que su aplicación requiere. Use un **DataSet** para hacer lo siguiente:

- Almacene datos en la memoria caché de la aplicación para poder manipularlos. Si solamente necesita leer los resultados de una consulta, el **DataReader** es la mejor elección.
- Utilizar datos de forma remota entre un nivel y otro o desde un servicio Web XML.
- Interactuar con datos dinámicamente, por ejemplo para enlazar con un control de formularios Windows Forms o para combinar y relacionar datos procedentes de varios orígenes.
- Realizar procesamientos exhaustivos de datos sin necesidad de tener una conexión abierta con el origen de datos, lo que libera la conexión para que la utilicen otros clientes.

Si no necesita la funcionalidad proporcionada por el **DataSet**, puede mejorar el rendimiento de su aplicación si utiliza el **DataReader** para devolver sus datos de sólo avance y de sólo lectura. Aunque el **DataAdapter** utiliza el **DataReader** para rellenar el contenido de un **DataSet**, al utilizar el **DataReader** puede mejorar el rendimiento porque no usará la memoria que utilizaría el **DataSet**, además de evitar el procesamiento necesario para crear y rellenar el contenido de **DataSet**.

### XML y ADO.NET

ADO.NET aprovecha la eficacia de XML para proporcionar acceso a datos sin mantener una conexión abierta. ADO.NET fue diseñado teniendo en cuenta las clases de XML incluidas en .NET Framework; ambos son componentes de una única arquitectura.

ADO.NET y las clases de XML incluidas en .NET Framework convergen en el objeto **DataSet**. El **DataSet** se puede llenar con datos procedentes de un origen XML, ya sea éste un archivo o una secuencia XML. El **DataSet** se puede escribir como XML compatible con el del Consorcio World Wide Web (W3C), incluyendo su esquema como esquema XSD (Lenguaje de definición de esquemas XML), independientemente del origen de los datos incluidos en el **DataSet**. Puesto que el formato nativo de serialización del **DataSet** es XML, es un medio excelente para mover datos de un nivel a otro, por lo que el **DataSet** es idóneo para utilizar datos y contextos de esquemas de interacción remota desde y hacia un servicio Web XML.

### Requisitos de la plataforma ADO.NET

Microsoft .NET Framework SDK (incluido ADO.NET) es compatible con Microsoft® Windows XP, Windows 2000, Windows NT 4 con Service Pack 6a, Windows Millennium Edition, Windows 98 y Windows CE.

En el siguiente ejemplo de código se muestra cómo incluir el espacio de nombres System.Data en las aplicaciones, para poder utilizar ADO.NET:

```
using System.Data;
```

### Proveedores de datos de .NET Framework

Un proveedor de datos de .NET Framework sirve para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente o se colocan en un DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, combinarlos con datos de varios orígenes o utilizarlos de forma remota entre niveles. Los proveedores de datos de .NET Framework son ligeros, de manera que crean un nivel mínimo entre el origen de datos y su código, con lo que aumenta el rendimiento sin sacrificar la funcionalidad.

En la tabla siguiente se muestran los proveedores de datos de .NET Framework que se incluyen en .NET Framework.

| Proveedor de datos de .NET Framework | Descripción |
|---|---|
| **Proveedor de datos de .NET Framework para SQL Server** | Proporciona acceso de datos para Microsoft SQL Server versión 7.0 o posterior. Utiliza el espacio de nombres System.Data.SqlClient. |
| **Proveedor de datos de .NET Framework para OLE DB** | Para orígenes de datos que se exponen mediante OLE DB. Utiliza el espacio de nombres System.Data.OleDb. |
| **Proveedor de datos de .NET Framework para ODBC** | Para orígenes de datos que se exponen mediante ODBC. Utiliza el espacio de nombres System.Data.Odbc. |
| **Proveedor de datos de .NET Framework para Oracle** | Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle y utiliza el espacio de nombres System.Data.OracleClient. |

El proveedor de datos de .NET Framework para ODBC y el proveedor de datos de .NET Framework para Oracle no se incluyeron originalmente en la versión 1.0 de .NET Framework. Si necesita utilizar el proveedor de datos de .NET Framework para ODBC o para Oracle, y está utilizando la versión 1.0 de .NET Framework, puede descargar dichos proveedores desde el sitio Data Access and Storage Developer Center. El espacio de nombres del proveedor de datos de .NET Framework para ODBC descargado es **Microsoft.Data.Odbc**. El espacio de nombres del proveedor de datos de .NET Framework para Oracle descargado es **System.Data.OracleClient**.

### Objetos principales de los proveedores de datos de .NET Framework

En la tabla siguiente se describen los cuatro objetos centrales que constituyen un proveedor de datos de .NET Framework.

| Objeto | Descripción |
|---|---|
| **Connection** | Establece una conexión a un origen de datos determinado. La clase base para todos los objetos **Connection** es DbConnection. |
| **Command** | Ejecuta un comando en un origen de datos. Expone **Parameters** y puede ejecutarse en el ámbito de un objeto **Transaction** de **Connection**. La clase base para todos los objetos **Command** es DbCommand. |
| **DataReader** | Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos. La clase base para todos los objetos **DataReader** es DbDataReader. |
| **DataAdapter** | Llena un **DataSet** y realiza las actualizaciones necesarias en el origen de datos. La clase base para todos los objetos **DataAdapter** es DbDataAdapter. |

Además de las clases principales citadas en la tabla anterior, los proveedores de datos de .NET Framework también incluyen las que se enumeran en la tabla siguiente.

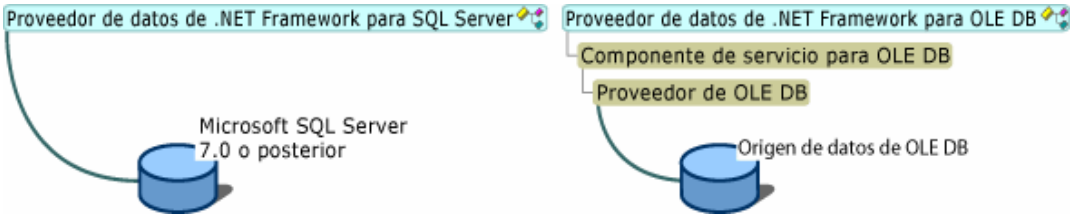
| Objeto | Descripción |
|---|---|
| **Transaction** | Permite incluir comandos en las transacciones que se realizan en el origen de datos. La clase base para todos los objetos **Transaction** es DbTransaction. |
| **CommandBuilder** | Un objeto auxiliar que genera automáticamente las propiedades de comando de un **DataAdapter** o que obtiene de un procedimiento almacenado información acerca de parámetros con la que puede rellenar la colección **Parameters** de un objeto **Command**. La clase base para todos los objetos **CommandBuilder** es DbCommandBuilder. |
| **ConnectionStringBuilder** | Un objeto auxiliar que proporciona un modo sencillo de crear y administrar el contenido de las cadenas de conexión utilizadas por los objetos **Connection**. La clase base para todos los objetos **ConnectionStringBuilder** es DbConnectionStringBuilder. |
| **Parameter** | Define los parámetros de entrada, salida y valores devueltos para los comandos y procedimientos almacenados. La clase base para todos los objetos **Parameter** es DbParameter. |
| **Exception** | Se devuelve cuando se detecta un error en el origen de datos. En el caso de que el error se detecte en el cliente, los proveedores de datos de .NET Framework inician una excepción de .NET Framework. La clase base para todos los objetos **Exception** es DbException. |
| **Error** | Expone la información relacionada con una advertencia o error devueltos por un origen de datos. |
| **ClientPermission** | Se proporciona para los atributos de seguridad de acceso a código de los proveedores de datos de .NET Framework. La clase base para todos los objetos **ClientPermission** es DBDataPermission. |

### Proveedor de datos de .NET Framework para SQL Server

El proveedor de datos de .NET Framework para SQL Server utiliza su propio protocolo para comunicarse con SQL Server. Es ligero y presenta un buen rendimiento porque está optimizado para tener acceso a SQL Server directamente, sin agregar una capa OLE DB u ODBC. En la siguiente ilustración se compara el proveedor de datos de .NET Framework para SQL Server y el proveedor de datos de .NET Framework para OLE DB. El proveedor de datos de .NET Framework para OLE DB se comunica con un origen de datos OLE DB tanto a través del componente de servicio OLE DB, que proporciona agrupación de conexiones y servicios de transacción, como del proveedor OLE DB correspondiente al origen de datos.

El proveedor de datos de .NET Framework para ODBC cuenta con una arquitectura similar a la del proveedor de datos de .NET Framework para OLE DB; por ejemplo, llama a un componente de servicio ODBC.

### Comparación entre el proveedor de datos de .NET Framework para SQL Server y el proveedor de datos de .NET Framework para OLE DB




Para utilizar el proveedor de datos de .NET Framework para SQL Server, debe tener acceso a SQL Server 7.0 o posterior. Las clases del proveedor de datos de .NET Framework para SQL Server se encuentran en el espacio de nombres **System.Data.SqlClient**. Para las versiones anteriores de SQL Server, utilice el proveedor de datos de .NET Framework para OLE DB con el proveedor OLE DB de SQL Server (SQLOLEDB).

El proveedor de datos de .NET Framework para SQL Server admite tanto transacciones locales como distribuidas. En el caso de las transacciones distribuidas, el proveedor de datos de .NET Framework para SQL Server se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma desde los servicios de componentes de Windows o System.Transactions.

En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.SqlClient** en sus aplicaciones.

```
using System.Data.SqlClient;
```

## Proveedor de datos de .NET Framework para OLE DB

El proveedor de datos de .NET Framework para OLE DB utiliza OLE DB nativo para permitir el acceso a datos mediante la interoperabilidad COM. El proveedor de datos de .NET Framework para OLE DB admite tanto transacciones locales como distribuidas. En el caso de las transacciones distribuidas, el proveedor de datos de .NET Framework para OLE DB se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma a través de los servicios de componentes de Windows 2000.

En la siguiente tabla se muestran los proveedores probados con ADO.NET.

| Controlador | Proveedor |
|---|---|
| SQLOLEDB | Proveedor OLE DB para SQL Server de Microsoft |
| MSDAORA | Proveedor OLE DB para Oracle de Microsoft |
| Microsoft.Jet.OLEDB.4.0 | Proveedor OLE DB para Microsoft Jet |

Las clases del proveedor de datos de .NET Framework para OLE DB se encuentran en el espacio de nombres **System.Data.OleDb**. En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.OleDb** en sus aplicaciones.

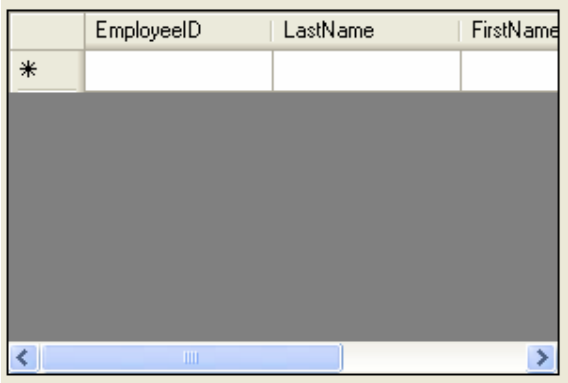
```
using System.Data.OleDb;
```

## DataGridView (Control, formularios Windows Forms)

El control **DataGridView** proporciona una forma eficaz y flexible de mostrar datos en formato de tabla. Puede utilizar el control **DataGridView** para mostrar vistas de sólo lectura de una cantidad pequeña de datos o puede ajustar su tamaño para mostrar vistas modificables de conjuntos muy grandes de datos.

Puede ampliar el control **DataGridView** de varias maneras para crear comportamientos personalizados en las aplicaciones. Por ejemplo, puede especificar mediante programación sus propios algoritmos de ordenación y crear sus propios tipos de celdas. Puede personalizar con facilidad la apariencia del control **DataGridView** seleccionando entre varias propiedades. Se pueden utilizar muchos tipos de almacenes de datos como origen de datos o el control **DataGridView** puede operar sin tener ningún origen de datos enlazado.

En los temas de esta sección se describen los conceptos y las técnicas que puede utilizar para crear características de **DataGridView** en las aplicaciones.




Con el control **DataGridView** puede mostrar y editar los datos en tablas a partir de numerosos tipos diferentes de orígenes de datos.

El enlace de datos al control **DataGridView** es sencillo e intuitivo y en muchos casos es tan fácil como establecer la propiedad **DataSource.** Cuando realiza el enlace a un origen de datos que contiene varias listas o tablas, establezca la propiedad **DataMember** en una cadena que especifica la lista o tabla a la que se va a enlazar.

Normalmente, se enlazará a un componente **BindingSource** y se enlazará el componente **BindingSource** a otro origen de datos o se rellenará con objetos de negocios. El componente **BindingSource** es el origen de datos preferido porque puede enlazarse a una amplia gama de orígenes de datos y puede resolver automáticamente muchos problemas de enlace de datos.

### Lo nuevo

Se ha diseñado el control **DataGridView** como una solución completa para mostrar datos en formato de tabla con formularios Windows Forms. Debería considerar utilizar el control **DataGridView** antes de otras soluciones, como DataGrid, cuando crea una nueva aplicación..

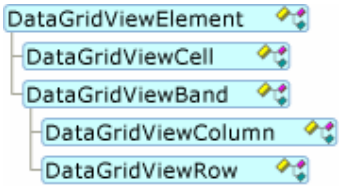
El control **DataGridView** puede funcionar en estrecha conjunción con el componente **BindingSource**. Este componente está diseñado para ser el origen de datos primario de un formulario. Puede administrar la interacción entre un control **DataGridView** y su origen de datos, sin tener en cuenta el tipo de origen de datos.

### Arquitectura del control DataGridView (formularios Windows Forms)

El control DataGridView y sus clases derivadas están diseñados para ser un sistema flexible y extensible para mostrar barras de herramientas y editar datos en formato de tabla. Todas estas clases están contenidas en el espacio de nombres System.Windows.Forms y se les denomina con el prefijo "DataGridView".

### Elementos de arquitectura

Las clases que acompañan a la clase **DataGridView** primaria derivan de DataGridViewElement.




La clase **DataGridViewElement** proporciona una referencia al control **DataGridView** primario y tiene una propiedad State que contiene un valor que representa una combinación de valores de la enumeración DataGridViewElementStates.

**DataGridViewElementStates**

La enumeración **DataGridViewElementStates** contiene los valores siguientes.

- None
- Frozen
- ReadOnly
- Resizable
- ResizableSet
- Selected
- Visible

Los valores de esta enumeración se pueden combinar con los operadores lógicos bit a bit, de modo que la propiedad **State** puede expresar más de un estado a la vez. Por ejemplo, un elemento **DataGridViewElement** puede ser simultáneamente **Frozen**, **Selected** y **Visible**.

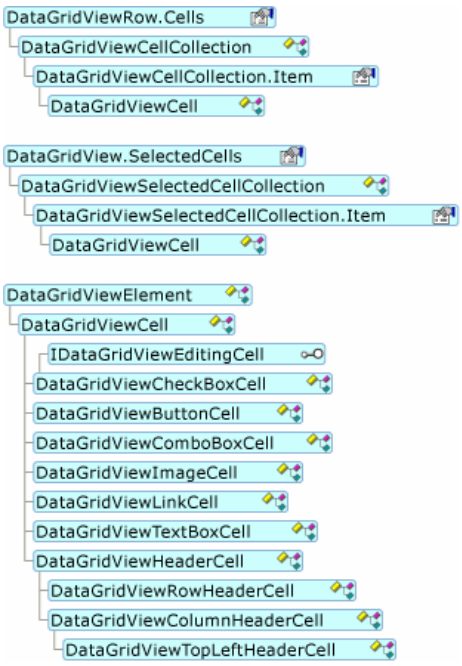
**Celdas y bandas**

El control **DataGridView** se compone de dos tipos fundamentales de objetos: celdas y bandas. Todas las celdas derivan de la clase base DataGridViewCell. Los dos tipos de bandas, DataGridViewColumn y DataGridViewRow, derivan de la clase base DataGridViewBand.

El control **DataGridView** interopera con varias clases, pero las más habituales son: **DataGridViewCell**, **DataGridViewColumn** y **DataGridViewRow**.

**DataGridViewCell**

**La celda es la unidad fundamental de interacción para** DataGridView**. La presentación se centra en las celdas y la entrada de datos se suele realizar a través de las celdas. Puede obtener acceso a las celdas utilizando la colección Cells de la clase** DataGridViewRow **y tener acceso a las celdas seleccionadas utilizando la colección SelectedCells del control** DataGridView**.**



El tipo **DataGridViewCell** es una clase base abstracta, de la que derivan todos los tipos de celdas. **DataGridViewCell** y sus tipos derivados no son controles de formularios Windows Forms, pero algunos alojan controles de formularios Windows Forms. Un control alojado controla normalmente cualquier función de edición admitida por una celda.

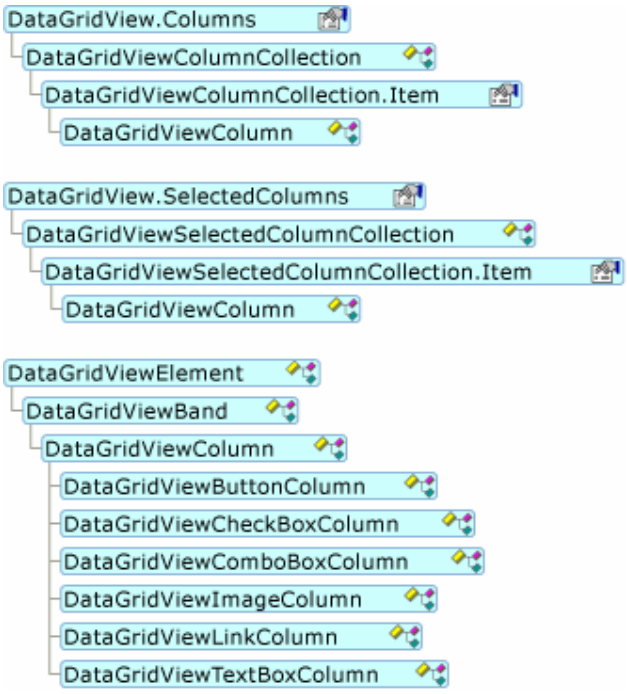
Los objetos **DataGridViewCell** no controlan su propia apariencia y las características de dibujo de la misma manera que los controles de formularios Windows Forms. En su lugar, **DataGridView** se encarga de la apariencia de los objetos **DataGridViewCell**. Puede afectar significativamente a la apariencia y comportamiento de celdas interactuando con las propiedades y eventos del control **DataGridView**. Si tiene requisitos especiales de personalizaciones que van más allá de los recursos del control **DataGridView**, puede implementar su propia clase que deriva de **DataGridViewCell** o de una de sus clases secundarias.

La lista siguiente muestra las clases derivadas de **DataGridViewCell**:

- DataGridViewTextBoxCell
- DataGridViewButtonCell
- DataGridViewLinkCell
- DataGridViewCheckBoxCell
- DataGridViewComboBoxCell
- DataGridViewImageCell
- DataGridViewHeaderCell
- DataGridViewRowHeaderCell
- DataGridViewColumnHeaderCell
- DataGridViewTopLeftHeaderCell
- Tipos de celda personalizados

**DataGridViewColumn**

El esquema del almacén de datos asociado del control **DataGridView** se expresa en las columnas del control **DataGridView**. Puede tener acceso a las columnas del control **DataGridView** utilizando la colección Columns. Puede tener acceso a las columnas seleccionadas utilizando la colección SelectedColumns.




Algunos de los tipos de celda clave tienen tipos de columna correspondientes. Éstos se derivan de la clase base **DataGridViewColumn**.

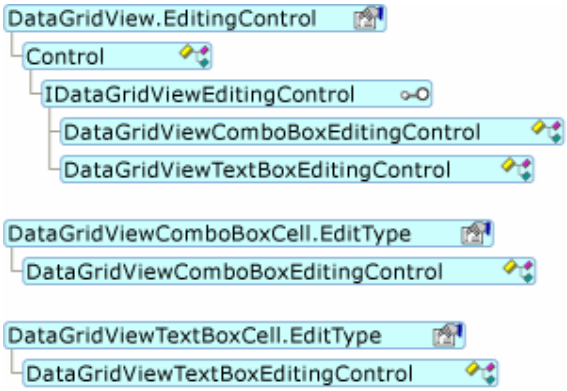
La lista siguiente muestra las clases derivadas de **DataGridViewColumn**:

- DataGridViewButtonColumn
- DataGridViewCheckBoxColumn
- DataGridViewComboBoxColumn
- DataGridViewImageColumn

- DataGridViewTextBoxColumn
- DataGridViewLinkColumn
- Tipos de columna personalizada

**Controles de edición DataGridView**

Las celdas que admiten funciones de edición avanzadas normalmente utilizan un control alojado que se deriva de un control de formularios Windows Forms. Estos controles también implementan la interfaz IDataGridViewEditingControl.




Se proporcionan los controles de edición siguientes con el control **DataGridView**:

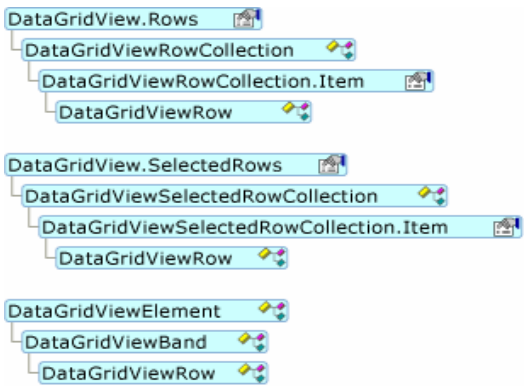
- DataGridViewComboBoxEditingControl
- DataGridViewTextBoxEditingControl

En la tabla siguiente se muestra la relación entre los tipos de celda, tipos de columna y controles de edición.

| Tipo de celda | Control alojado | Tipo de columna |
|---|---|---|
| **DataGridViewButtonCell** | no disponible | **DataGridViewButtonColumn** |
| **DataGridViewCheckBoxCell** | no disponible | **DataGridViewCheckBoxColumn** |
| **DataGridViewComboBoxCell** | **DataGridViewComboBoxEditingControl** | **DataGridViewComboBoxColumn** |
| **DataGridViewImageCell** | no disponible | **DataGridViewImageColumn** |
| **DataGridViewLinkCell** | no disponible | **DataGridViewLinkColumn** |
| **DataGridViewTextBoxCell** | **DataGridViewTextBoxEditingControl** | **DataGridViewTextBoxColumn** |

**DataGridViewRow**

La clase **DataGridViewRow** muestra campos de datos de un registro del almacén de datos al que se asocia el control **DataGridView**. Puede tener acceso a las filas del control **DataGridView** utilizando la colección Rows. Puede tener acceso a las filas seleccionadas utilizando la colección SelectedRows.



Puede derivar sus propios tipos de la clase **DataGridViewRow**, aunque esto normalmente no será necesario. El control **DataGridView** tiene varios eventos relacionados con fila y propiedades para personalizar el comportamiento de los objetos **DataGridViewRow**.

Si habilita la propiedad AllowUserToAddRows del control **DataGridView**, aparecerá en la última fila una fila especial para agregar nuevas filas. Esta fila forma parte de la colección **Rows**, pero tiene funcionalidad especial que puede requerir su atención.

## BINDINGSOURCE 


Encapsula un origen de datos para enlazarlo a controles.

El componente BindingSource sirve para dos propósitos. Primero, proporciona una capa de direccionamiento indirecto al enlazar los controles de un formulario a los datos. Esto se lleva a cabo enlazando el componente **BindingSource** a su origen de datos y enlazando a continuación los controles del formulario al componente **BindingSource**. Toda la demás interacción con los datos, incluido el desplazamiento, la ordenación, el filtrado y la actualización, se lleva a cabo con llamadas al componente **BindingSource**.

En segundo lugar, el componente **BindingSource** puede actuar como un origen de datos con establecimiento inflexible de tipos. Al agregar un tipo al componente **BindingSource** con el método Add se crea una lista de ese tipo.

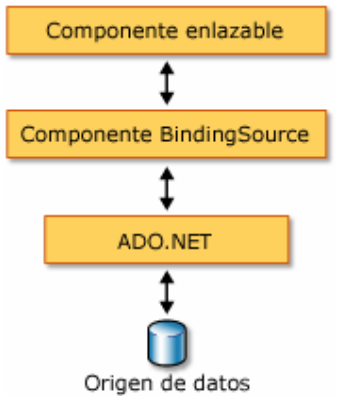
### Información general sobre el componente BindingSource

El componente BindingSource está diseñado para simplificar el proceso de enlazar controles a un origen de datos subyacente. El componente **BindingSource** actúa como canalización y como origen de datos para otros controles a los que enlazarlo. Proporciona una abstracción de la conexión de datos de su formulario a la vez que se pasan mediante comandos a la lista subyacente de datos. Además, puede agregar los datos directamente a él, para que el componente funcione como un origen de datos.

### El componente BindingSource como intermediario

El componente **BindingSource** actúa como el origen de datos para algunos o todos los controles del formulario. En Visual Studio, **BindingSource** se puede enlazar a un control por medio de la propiedad **DataBindings** a la que se tiene acceso desde la ventana **Propiedades**.

Puede enlazar un componente **BindingSource** tanto a orígenes de datos simples, una única propiedad de un objeto o una colección básica como ArrayList, como a orígenes de datos complejos, como una tabla de base de datos. El componente **BindingSource** actúa como intermediario que proporciona enlaces y servicios de administración de moneda. Puede enlazar un componente **BindingSource** a un origen de datos complejo estableciendo sus propiedades DataSource y DataMember a la base de datos y a la tabla respectivamente, en tiempo de diseño y en tiempo de ejecución. La siguiente ilustración muestra dónde encaja el componente **BindingSource** en la arquitectura de enlace de datos existente.




### El componente BindingSource como origen de datos

Si comienza agregando elementos al componente **BindingSource** sin haber especificado primero una lista a la que enlazarlo, el componente actuará como origen de datos de estilo de lista y aceptará los elementos que se han agregado.

Además, puede escribir código para proporcionar funcionalidad "AddNew" personalizada por medio del evento AddingNew, que se provoca al llamar al método AddNew antes de agregar el elemento a la lista.

## Desplazamiento

Para los usuarios que necesiten explorar los datos de un formulario, el componente BindingNavigator le permite explorar y manipular datos en coordinación con un componente **BindingSource**.

### Manipulación de datos

**BindingSource** funciona como un CurrencyManager para todos sus enlaces y, por tanto, puede proporcionar acceso a la información de posición y de moneda en relación con el origen de datos. La tabla siguiente muestra los miembros que el componente **BindingSource** proporciona para obtener acceso y manipular los datos subyacentes.

| Miembro | Descripción |
|---|---|
| Propiedad Current | Obtiene el elemento activo del origen de datos. |
| Propiedad Position | Obtiene o establece la posición actual en la lista subyacente. |
| Propiedad List | Obtiene la lista que es la evaluación de **DataSource** y **DataMember**. Si no se establece **DataMember**, devuelve la lista especificada por **DataSource**. |
| Método Insert | Inserta un elemento en la lista en el índice especificado. |
| Método RemoveCurrent | Quita el elemento actual de la lista. |
| Método EndEdit | Aplica los cambios pendientes al origen de datos subyacente. |
| Método CancelEdit | Cancela la operación de edición actual. |
| Método **AddNew** | Agrega un nuevo elemento a la lista subyacente. Si el origen de datos implementa IBindingList y devuelve un elemento del evento **AddingNew**, agrega este elemento. De lo contrario, la solicitud se pasa al método AddNew de la lista. Si la lista subyacente no es una **IBindingList**, el elemento se crea automáticamente a través de su constructor predeterminado público. |

## Ordenar y filtrar

Normalmente, debería trabajar con una vista del origen de datos ordenada o filtrada. La tabla siguiente muestra los miembros que proporciona el origen de datos del componente **BindingSource**.

| Miembro | Descripción |
|---|---|
| Propiedad Sort | Si el origen de datos es una **IBindingList**, obtiene o establece un nombre de columna utilizado para ordenar y la información del criterio de ordenación. Si el origen de datos es una IBindingListView y admite la ordenación avanzada, obtiene varios nombres de columna utilizados para ordenar, así como el criterio de ordenación. |
| Propiedad Filter | Si el origen de datos es una **IBindingListView**, obtiene o establece la expresión utilizada para filtrar las filas que se ven. |

## BindingNavigator (Control, formularios Windows Forms)

El control **BindingNavigator** es la interfaz de usuario (IU) de exploración y manipulación para controles que se enlazan a datos. El control **BindingNavigator** permite a los usuarios desplazarse por los datos y manipularlos en formularios Windows Forms.

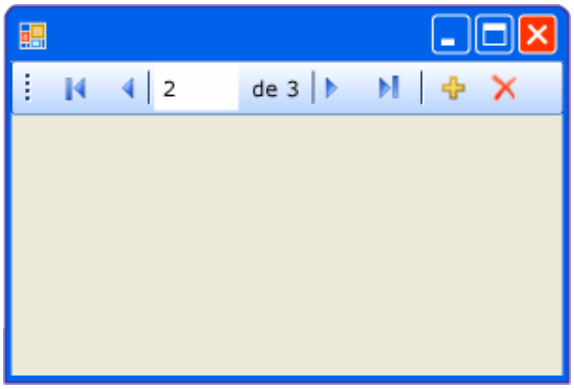
En los temas de esta sección se proporciona información general del control **BindingNavigator** y se ofrecen instrucciones paso a paso sobre cómo utilizar los datos de desplazamiento del control y desplazarse por un DataSet.

Puede utilizar el control BindingNavigator para crear un medio estandarizado con el que los usuarios busquen y cambien los datos en un formulario Windows Forms. **BindingNavigator** se utiliza frecuentemente con el componente

BindingSource para permitir a los usuarios a desplazarse a través de los registros de datos en un formulario e interactuar con los registros.

## Cómo funciona BindingNavigator

El control **BindingNavigator** está compuesto de ToolStrip con una serie de objetos ToolStripItem para las acciones más comunes relacionadas con datos: agregarlos, eliminarlos y explorarlos. De manera predeterminada, el control **BindingNavigator** contiene estos botones estándar. La captura de pantalla siguiente muestra el control **BindingNavigator** en un formulario.




La tabla siguiente enumera los controles y describe sus funciones.

| Control | Función |
|---|---|
| Botón AddNewItem | Inserta una nueva fila en el origen de datos subyacente. |
| Botón DeleteItem | Elimina la fila actual del origen de datos subyacente. |
| Botón MoveFirstItem | Desplaza al primer elemento del origen de datos subyacente. |
| Botón MoveLastItem | Desplaza al último elemento del origen de datos subyacente. |
| Botón MoveNextItem | Desplaza al siguiente elemento del origen de datos subyacente. |
| Botón MovePreviousItem | Desplaza al elemento anterior del origen de datos subyacente. |
| Cuadro de texto PositionItem | Devuelve la posición actual dentro del origen de datos subyacente. |
| Cuadro de texto CountItem | Devuelve el número total de elementos del origen de datos subyacente. |

Para cada control de esta colección, existe un miembro equivalente del componente T:System.Windows.Forms.BindingSource que proporciona la misma funcionalidad mediante programación. Por ejemplo, el botón **MoveFirstItem** corresponde al método MoveFirst del componente **BindingSource**, el botón **DeleteItem** corresponde al método RemoveCurrent, y así sucesivamente.

Si los botones predeterminados no se ajustan a su aplicación o si necesita botones adicionales para la compatibilidad con otros tipos de funcionalidad, puede proporcionar sus propios botones **ToolStrip**.

## TableAdapter

Los TableAdapters comunican la aplicación con una base de datos. Más específicamente, un TableAdapter se conecta con una base de datos, ejecuta consultas o procedimientos almacenados, y devuelve una nueva tabla de datos rellena con los datos devueltos o rellena una DataTable existente con los datos devueltos. Los TableAdapters también se utilizan para devolver los datos actualizados desde la aplicación a la base de datos.

Los usuarios de versiones anteriores de Visual Studio pueden pensar en un TableAdapter como un DataAdapter que lleva integrado un objeto de conexión y la capacidad de contener varias consultas. Cada consulta agregada a un TableAdapter se expone como un método público al que se llama simplemente como a cualquier otro método o función de un objeto.

Además de la funcionalidad estándar del control **DataAdapter**, los objetos TableAdapter proporcionan métodos con tipo adicionales que encapsulan consultas que comparten un esquema común con el control **DataTable** con tipo asociado. Dicho de otra forma, puede tener tantas consultas como desee en un TableAdapter, siempre y cuando devuelvan datos que cumplan el mismo esquema.

En la versión anterior de Visual Studio, Adaptadores de datos ADO.NET se utilizó para comunicar una aplicación con una base de datos. Aunque los adaptadores de datos siguen siendo un componente principal de Proveedores de datos de .NET Framework, los objetos TableAdapter son componentes generados por el diseñador que mejoran la funcionalidad de los controles **DataAdapter**. Estos objetos suelen contener los métodos Fill y Update para obtener y actualizar los datos en una base de datos.

Los TableAdapter se crean con el **Diseñador de DataSet** dentro de los conjuntos de datos con establecimiento inflexible de tipos. Con Asistente para la configuración de orígenes de datos puede crear objetos TableAdapter durante la creación de un nuevo conjunto de datos. También puede crearlos en conjuntos de datos existentes con el Asistente para la configuración de TableAdapter o arrastrando los objetos de base de datos del **Explorador de servidores** al **Diseñador de DataSet**.

Aunque los objetos TableAdapter se diseñan con el **Diseñador de DataSet**, las clases TableAdapter generadas no se generan como clases anidadas de DataSet. Se encuentran en un espacio de nombres separado y específico para cada conjunto de datos. Por ejemplo, si tiene un conjunto de datos denominado NorthwindDataSet, los TableAdapter asociados al control **DataTable** en NorthwindDataSet estarían en el espacio de nombres NorthwindDataSetTableAdapters. Para tener acceso a un TableAdapter determinado mediante programación, debe declarar una nueva instancia del TableAdapter. Por ejemplo

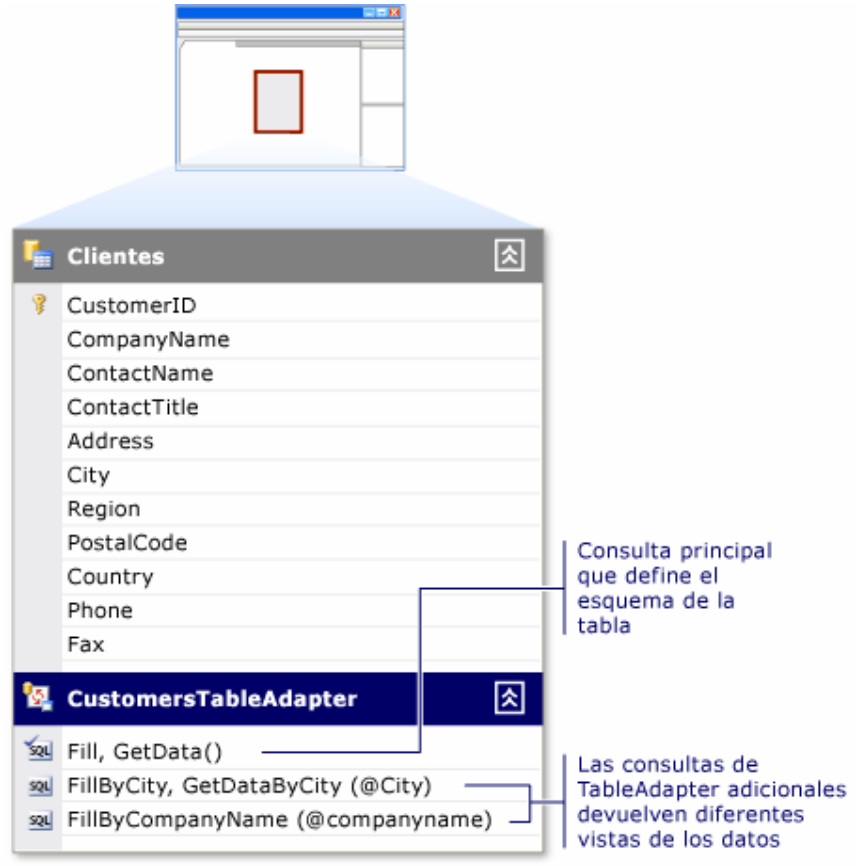
```
NorthwindDataSet northwindDataSet = new NorthwindDataSet();
NorthwindDataSetTableAdapters.CustomersTableAdapter customersTableAdapter =
    new NorthwindDataSetTableAdapters.CustomersTableAdapter();
customersTableAdapter.Fill(northwindDataSet.Customers);
```

### Esquema de DataTable asociado

Al crear un objeto TableAdapter, se utiliza la consulta inicial o el procedimiento almacenado para definir el esquema de **DataTable** asociado al TableAdapter. Esta consulta inicial o procedimiento almacenado se ejecuta llamando al método principal Fill del TableAdapter (que rellena la **DataTable** asociada al TableAdapter). Cualquier cambio realizado en la consulta principal del TableAdapter se refleja en el esquema de la tabla de datos asociada. Por ejemplo, al quitar una columna de la consulta principal, se quita la columna de la tabla de datos asociada. Si alguna consulta adicional del TableAdapter utiliza instrucciones SQL que devuelven columnas que no están en la consulta principal, el diseñador intentará sincronizar los cambios de columna entre la consulta principal y cualquier consulta adicional.

### Comandos de actualización de TableAdapter

La funcionalidad de actualización de un objeto TableAdapter depende de la cantidad de información disponible, en función de la consulta principal proporcionada en el Asistente de TableAdapter. Por ejemplo, los TableAdapter configurados para obtener valores de varias tablas (consultas JOIN), valores escalares, vistas o los resultados de funciones de agregado no se crean inicialmente con la capacidad de enviar actualizaciones a la base de datos subyacente. Sin embargo, puede configurar manualmente los comandos INSERT, UPDATE y DELETE en la ventana **Propiedades**.




A diferencia de los adaptadores de datos estándar, los TableAdapter pueden contener varias consultas que rellenan las tablas de datos asociadas. Puede definir tantas consultas para un TableAdapter como requiera la aplicación, con tal de que cada consulta devuelva datos que cumplan el mismo esquema que la tabla de datos asociada. De esta forma se habilita la carga de datos que satisface distintos criterios. Por ejemplo, si la aplicación contiene una tabla de clientes, puede crear una consulta para rellenar la tabla con todos los clientes cuyo nombre comience con una letra determinada y otra consulta para rellenar la tabla con todos los clientes del mismo estado o provincia. Para rellenar una tabla Customers con clientes de un estado concreto, puede crear una consulta FillByState que toma un parámetro para el valor del estado: SELECT * FROM Customers WHERE State = @State. Ejecute la consulta llamando al método FillByState y pasando un valor del parámetro como: CustomerTableAdapter.FillByState("WA"). Además de consultas que devuelven datos del mismo esquema como tabla de datos del TableAdapter, también puede agregar consultas que devuelvan valores escalares (únicos). Por ejemplo, la creación de una consulta que devuelve un recuento de clientes (SELECT Count(*) From Customers) es una consulta válida para CustomersTableAdapter, aunque los datos devueltos no cumplan el esquema de la tabla.

## Métodos y propiedades de TableAdapter

La clase TableAdapter no forma parte de .NET Framework y como tal, no puede buscarla en la documentación ni en el **Examinador de objetos**. Se crea en tiempo de diseño cuando utiliza uno de los asistentes anteriormente mencionados. El nombre asignado a un TableAdapter en el momento de su creación se basa en el nombre de la tabla con la que está trabajando. Por ejemplo, si se crea un TableAdapter basado en una tabla de una base de datos denominada Orders, los TableAdapter se denominan en consecuencia OrdersTableAdapter. Se puede cambiar el nombre de clase del TableAdapter utilizando la propiedad **Name** en el **Diseñador de DataSet**.

A continuación se muestran los métodos y propiedades de TableAdapter más utilizados:

| Miembro | Descripción |
|---|---|
| TableAdapter.Fill | Rellena la tabla de datos asociada del TableAdapter con los resultados del comando SELECT del TableAdapter. Para obtener más información, vea Cómo: Llenar un conjunto de datos con datos. |
| TableAdapter.Update | Devuelve los cambios a la base de datos. Para obtener más información, vea Cómo: Actualizar datos utilizando un TableAdapter. |

| | |
|---|---|
| TableAdapter.GetData | Devuelve una nueva **DataTable** rellena con datos. |
| TableAdapter.Insert | Crea una nueva fila en la tabla de datos. Para obtener más información, vea Cómo: Agregar filas a un DataTable. |
| TableAdapter.ClearBeforeFill | Determina si se vacía una tabla de datos antes de llamar a uno de los métodos Fill. |

## Las siguientes características son nuevas en ADO.NET versión 2.0.

### Proveedores administrados

**Enumeración de servidores**

- Ofrece compatibilidad con la enumeración de instancias activas de Microsoft SQL Server 2000 y posterior.

**Procesamiento asincrónico**

- Permite realizar operaciones asincrónicas de base de datos mediante una API modelada después del modelo asincrónico que utiliza .NET Framework. .

**Varios conjuntos de resultados activos (MARS)**

- Permite que las aplicaciones tengan más de un SqlDataReader abierto en una conexión cuando cada instancia de **SqlDataReader** se inicia desde un comando distinto.

**Operaciones de copia masiva**

- Permite realizar inserciones masivas rápidas mediante el proveedor de datos de .NET para SQL Server.

**Nuevos tipos de datos máximos de SQL Server**

- Ofrece compatibilidad con los tipos de datos **varchar(max)**, **nvarchar(max)**, **varbinary(max)** en SQL Server 2005.

**Tipos definidos por el usuario de SQL Server**

- Ofrece compatibilidad con tipos de datos definidos por el usuario (UDT) en SQL Server 2005.

**Notificaciones de SQL Server**

- Permite que las aplicaciones .NET Framework envíen un comando a SQL Server y soliciten que se genere una notificación si la ejecución del mismo comando fuera a producir conjuntos de resultados diferentes de los inicialmente recuperados.

**Transacciones con aislamiento de instantáneas de SQL Server**

- Ofrece compatibilidad con el aislamiento de instantáneas, un mecanismo de SQL Server 2005 diseñado para reducir el bloqueo en aplicaciones OLTP.

**Reflejo de bases de datos en SQL Server 2005**

- Ofrece compatibilidad con el reflejo de bases de datos en SQL Server 2005 con una nueva sintaxis de cadena de conexión para especificar un servidor asociado de conmutación por error.

**Estadísticas de proveedor**

- Ofrece compatibilidad con la recuperación de estadísticas de tiempo de ejecución en SQL Server 2005. Actualmente hay disponibles 21 contadores diferentes desde el proveedor de .NET para SQL Server.

**Cambio de contraseña en SQL Server 2005**

- Permite que las aplicaciones .NET Framework cambien la contraseña de una cuenta de usuario sin necesidad de que intervenga el administrador.

**Procesamiento por lotes**

- Mejora el rendimiento de la aplicación mediante la reducción del número de viajes de ida y vuelta a la base de datos al aplicar las actualizaciones desde el DataSet.

**Seguimiento**

- ADO.NET 2.0 presenta una nueva funcionalidad integrada de seguimiento de datos que admiten los proveedores de datos de .NET.

**Confianza parcial**

- Todos los proveedores de datos de Microsoft se admiten ahora en entornos de confianza parcial.

**Control de agrupamiento de conexiones**

- ADO.NET 2.0 presenta dos nuevos métodos para borrar el grupo de conexión: ClearAllPools y ClearPool.

**Compatibilidad con el tipo de datos XML de SQL Server**

- SQL Server 2005 tiene un nuevo tipo de datos XML y System.Data.SqlClient le ofrece una robusta compatibilidad en el cliente. Los valores XML se exponen mediante el marco System.Xml para conseguir la completa integración con el modelo de programación .NET.

**Integración con la optimización de transacciones del sistema y transacciones promocionadas para SQL Server 2005**

- .NET 2.0 incluye un nuevo marco de transacciones, accesible a través del espacio de nombres System.Transactions. Al combinar **System.Transactions** para la administración de transacciones distribuidas, **System.Data.SqlClient** para el acceso a la base de datos y SQL Server 2005 como servidor, es posible optimizar las transacciones distribuidas de modo que el costo extra de convertirlas en "distribuidas" sólo se realice cuando las transacciones son realmente necesarias.

**Clases desconectadas**

**Mejoras de DataSet**

- El nuevo **DataTableReader** presenta el contenido de un **DataSet** o una DataTable con formato de uno o más conjuntos de resultados de sólo lectura y de sólo avance.

**Mejoras de DataSet**

- Un nuevo motor de índice mejora el rendimiento de las operaciones de inserción, eliminación y modificación para los objetos **DataTable** y DataRow. Es necesario definir un índice o una clave primaria en la tabla base.

**Serialización binaria del DataSet**

- La nueva opción permite la serialización de un **DataSet** y de una **DataTable** en formato binario cuando se utilizan transportes binarios a través del servicio remoto. En la mayoría de los casos, el resultado es una enorme mejora en el rendimiento y una reducción significativa del uso de memoria y CPU cuando se utilizan objetos **DataSet**/**DataTable** en aplicaciones que emplean el servicio remoto para conectarse a niveles diferentes.

**DataTable como objeto independiente**

- Muchos de los métodos que en las versiones anteriores sólo estaban disponibles en el **DataSet** se encuentran ahora también disponibles en la **DataTable** (por ejemplo, **ReadXml** y **WriteXml**). Además, es posible serializar una **DataTable** por sí sola, así que ya no es necesario tener un **DataSet** con una única tabla simplemente para exponer la tabla a través de servicios Web o de cualquier otro mecanismo que requiera serialización.

**Creación de una DataTable desde una DataView**

- Ahora puede crear una **DataTable** desde una DataView. La nueva **DataTable** tendrá el mismo conjunto de filas que la **DataView** actual. De manera opcional, las columnas de la nueva **DataTable** pueden ser un subconjunto de las columnas **DataView**. La filas pueden ser todas las filas o sólo filas distintas.

**Mejoras del motor de inferencia de esquemas**

- El motor de inferencia de esquemas se ha mejorado con la deducción de tipos cuando existe información suficiente; además, es bastante más rápido.

**Tablas completas de espacio de nombres**

- Ahora, un **DataSet** puede admitir tablas que tienen el mismo TableName pero diferente Namespace, gracias a un mecanismo que evita el conflicto entre nombres.

**Nuevas capacidades de carga de DataTable**

- El nuevo método **Load** para **DataTables** y **DataSets** puede transmitir en secuencia un **DataReader** directamente a una **DataTable**. El método **Load** presenta también nuevas opciones para el comportamiento de carga que amplían la funcionalidad disponible a través del **DataAdapter**.

**Control de estado de fila**

- Los nuevos métodos SetAdded y SetModified permiten que las aplicaciones manipulen explícitamente el estado de las filas de **DataSet** y **DataTable**.

**Compatibilidad mejorada con tipos definidos personalizados**

- La **DataTable** ofrece una mayor compatibilidad con tipos personalizados y definidos por el usuario. Se ha ampliado la serialización XML para la compatibilidad con el polimorfismo. Por ejemplo, una columna Person puede almacenar una instancia Employee que se serialice y deserialice a y desde XML sin perder la fidelidad de ningún tipo. Además, ahora los usuarios pueden implementar las nuevas interfaces de seguimiento de cambios que permiten que la **DataTable** realice el seguimiento de los cambios en tales objetos.

**Mejoras de XML/XSD**

- Se ha agregado compatibilidad con la lectura y escritura de tipos simples en XSD. Se ha agregado compatibilidad con la lectura y escritura de varios esquemas en una secuencia. Se ha agregado compatibilidad con la lectura y escritura de XSD/XML cuando una sola tabla o un único elemento XML está anidado en dos o más tablas primarias.

**Mejoras de DataRow**

- El nuevo IndexOf se puede utilizar para buscar la posición de una fila en colecciones **DataTable.Rows**.

**Cambio de posición de las columnas en DataTable**

- El nuevo método **SetOrdinal** permite a los usuarios cambiar la posición de la columna en una **DataTable**.

**Mejoras en el rendimiento de DataView**

- La **DataView** de ADO.NET 2.0 utiliza un nuevo algoritmo de actualización que proporciona una mejora significativa en el mantenimiento de **DataView**.

**Expresiones de DataColumn**

- Una expresión de una DataColumn puede incluir columnas de la misma **DataTable**. La expresión se puede actualizar y el valor de columna se mantiene.

**Serialización de esquemas para DataSets con información de tipos**

- Los **DataSets** con información de tipos pueden excluir el componente de esquema de la serialización si se establece la enumeración de **SchemaSerializationMode** en **ExcludeSchema**.

**API independientes del proveedor**

**Enumeración del proveedor de datos de .NET**

- Ofrece compatibilidad con la enumeración de los proveedores de datos instalados.

**API independiente del proveedor**

- Las mejoras en el espacio de nombres System.Data.Common proporcionan acceso a través de una única API a las bases de datos de varios proveedores.

**Descubrimiento de esquemas**

- El descubrimiento de esquemas permite que las aplicaciones soliciten a los proveedores administrados que busquen y devuelvan información acerca del esquema de la base de datos a la que está conectada una determinada conexión. Los diferentes elementos de esquema de base de datos, como tablas, columnas y procedimientos almacenados, se exponen a través de los métodos **GetSchema** de la clase **Connection** de cada proveedor.