DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: *CC035 VISUAL BASIC AVANZADO*
FECHA: *5 al 16 de junio del 2000*

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
FABIAN OLIVER SUAREZ				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso_____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

2. Medio a través del cual se enteró del curso:

Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

3. ¿Qué cambios sugeriría al curso para mejorarlo?

_____

_____

_____

_____

_____

_____

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ☐ NO ☐

5.¿Qué cursos sugiere que imparta la División de Educación Continua?

_____

_____

_____

_____

_____

_____

6. Otras sugerencias:

_____

_____

_____

_____

_____

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

MATERIAL DIDACTICO

DEL CURSO

VISUAL BASIC AVANZADO

Junio del 2000

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

Visual Basic Avanzado

OBJETIVO

Al finalizar el curso el participante manejara la programación orientada a eventos, haciendo uso de las herramientas que proporciona VISUAL BASIC para el desarrollo de aplicaciones usando esta filosofía de programación.
El participante utilizará los conocimientos del VISUAL BASIC básico para manejar archivos de base de datos y construir aplicaciones completas.

Duración: 40 hrs

Temario

1. Repaso de Microsoft Visual Basic
   La filosofía de desarrollo de Visual Basic
   Diseño de Interfaces de Usuario
   ¿Cómo es almacenado el código?
   Escribiendo código de programación
   Validación de entradas
   Compilación de programas y generación de archivos ejecutables

2. Ejecución y Depuración
   Puntos de control en la ejecución
   Añadiendo vistas de depuración
   Ejecución paso a paso

3. Controles y Objetos para el manejo de Base de Datos
   Características de los objetos relacionados a datos
   El objeto Database
   El acceso a base de datos utilizando Visual Basic
   El objeto data control sus métodos y propiedades

4. Manejo de Base de Datos de Access
   Arquitectura de las Base de Datos de Access
   Uso de Base de Datos de Access con Visual Basic
   Uso de otros tipos de Base de Datos soportadas por Visual Basic
   Creación de Recordset y Snapshot mediante consultas
   Navegación de un Recordset

5. Desarrollo Avanzado de Base de Datos.
   Manejo de integridad referencial
   Violaciones a los datos
   Elementos multiusuario
   El esquema ODBD

6. Creación de Reportes Impresos
   Diseño de reportes
   Impresión de reportes tipo Recordset
   Uso del método Print

7. Uso de librerías de Ligas Dinámicas
   El propósito de las DLL´s
   Implementando llamadas a funciones en las DLL´s

8. Creación de Archivos de Ayuda para Windows
   Creación del texto de ayuda
   Creación del archivo de proyecto
   Creación del archivo de ayuda
   Integración de la ayuda al programa
   Creación de discos de distribución

Notas

El curso se manejará tipo taller. Habrá una breve explicación teórica para recordar los conceptos y el alumno desarrollara un programa utilizando dichos conceptos. La idea del curso es que al finalizarlo el alumno logre construir una pequeña aplicación de acuerdo a los alcances y limitaciones del curso.

Una aplicación bajo Windows presenta típicamente todas las opciones en una pantalla (en la forma de objetos visuales). De esta forma, esto representa una forma completamente nueva de programación - orientada a eventos y orientada a objetos. Es decir, el programador no es completamente responsable del flujo del programa - el usuario es el responsable. El usuario selecciona una de las opciones posibles, y le corresponde al programa responder correctamente.

Los formularios son la base para crear la interfaz de una aplicación. Puede usar formularios para agregar ventanas y cuadros de diálogo a la aplicación. También puede usarlos como contenedores de elementos que no son parte visible de la interfaz de la aplicación. Por ejemplo, puede tener un formulario en su aplicación que sirva como contenedor para gráficos que quiera presentar en otros formularios.

El primer paso para generar una aplicación de Visual Basic consiste en crear los formularios que van a ser la base de la interfaz de su aplicación. Después dibuje los objetos que van a componer la interfaz en los formularios que ha creado. Para esta primera aplicación, usaremos dos controles del cuadro de herramientas.

1.3 Como es Almacenado el Código

Las siguientes secciones describen los diferentes tipos de archivos y objetos que se pueden incluir en un proyecto.

Módulos de formulario

Los módulos de formulario (extensión de nombre de archivo .frm) pueden contener texto descriptivo del formulario y sus controles, incluyendo los valores de sus propiedades. También pueden contener declaraciones de formulario de constantes, variables y procedimientos externos, así como procedimientos de evento y procedimientos generales.

Módulos de clase

Los módulos de clase (extensión de nombre de archivo .cls) son similares a los módulos de formulario, excepto en que no tienen interfaz de usuario visible. Puede usar módulos de clase para crear sus propios objetos, incluyendo código para métodos y propiedades.

Módulos estándar

Los módulos estándar (extensión de nombre de archivo .bas) pueden contener declaraciones públicas o a nivel de módulo de tipos, constantes, variables, procedimientos externos y procedimientos públicos.

Archivos de recursos

Los archivos de recursos (extensión de nombre de archivo .res) contienen mapas de bits, cadenas de texto y otros datos que se pueden modificar sin volver a modificar el código. Por ejemplo, si piensa localizar su aplicación a un idioma extranjero, puede guardar todas las cadenas de texto de la interfaz de usuario y los mapas de bits en un archivo de recursos, y simplemente traducir el archivo de recursos en vez de la aplicación completa. Un proyecto sólo puede contener un archivo de recursos.

Documentos ActiveX

Los documentos ActiveX (.dob) son similares a los formularios, pero se muestran en un explorador de Internet como Internet Explorer. Las ediciones Profesional y Empresarial de Visual Basic son capaces de crear documentos ActiveX.

Módulos de controles de usuario y de páginas de propiedades

Los módulos de controles de usuario (.ctl) y de páginas de propiedades (.pag) son similares a los formularios, pero se usan para crear controles ActiveX y las páginas de propiedades asociadas para mostrar propiedades en tiempo de diseño. Las versiones Profesional y Empresarial de Visual Basic pueden crear controles ActiveX.

Componentes

Además de archivos y módulos, también es posible agregar otro tipo de componentes a un proyecto.

Controles ActiveX

Los controles ActiveX (extensión de nombre de archivo .ocx) son controles opcionales que se pueden agregar al cuadro de herramientas y se pueden usar en formularios. Cuando instala Visual Basic, los archivos que contienen los controles incluidos en Visual Basic se copian a un directorio común (el subdirectorio \Windows\System en Windows 95). Existen controles ActiveX adicionales disponibles en diversas fuentes. También puede crear sus propios controles mediante las ediciones Profesional y Empresarial de Visual Basic.

Objetos insertables

Los *objetos insertables*, como un objeto Hoja de cálculo de Microsoft Excel, son componentes que se pueden usar como bloques para generar soluciones integradas. Una *solución integrada* puede contener datos en diferentes formatos, como hojas de cálculo, mapas de bits y texto, creados por diferentes aplicaciones.

Referencias

También puede agregar referencias a componentes ActiveX externos que se pueden usar en la aplicación. Para asignar referencias se usa el cuadro de diálogo **Referencias**, al que se tiene acceso mediante el comando **Referencias** del menú **Proyecto**.

Diseñadores ActiveX

Los diseñadores ActiveX son herramientas para diseñar clases a partir de las cuales es posible crear objetos. La interfaz de diseño para formularios es el diseñador predeterminado. Puede haber disponibles otros diseñadores adicionales desde otros orígenes.

Controles estándar

Los controles estándar los proporciona Visual Basic. Los controles estándar, como **CommandButton** (botón de comando) o **Frame** (marco), siempre están incluidos en el cuadro de herramientas, al contrario de lo que ocurre con los controles ActiveX y los objetos insertables, que se pueden agregar y quitar del cuadro de herramientas.

1.4 Escribiendo el Código de Programación

El código en Visual Basic se almacena en módulos. Hay tres tipos de módulos: de formulario, estándar y de clase.

Las aplicaciones sencillas pueden consistir en un único formulario y todo el código de la aplicación reside en ese módulo de formulario. A medida que sus aplicaciones vayan creciendo y siendo más sofisticadas, agregará formularios adicionales. A veces tendrá código común que deseará ejecutar en varios formularios. No querrá duplicar el código en ambos formularios, por lo que creará un módulo independiente que contenga un procedimiento que ejecuta el código común. Este módulo independiente debe ser un módulo estándar. Con el tiempo, puede construir una biblioteca de módulos que contenga los procedimientos compartidos.

Cada módulo estándar, de clase y de formulario puede contener lo siguiente:

Declaraciones. Puede colocar declaraciones de constantes, tipos, variables y procedimientos de bibliotecas de vínculos dinámicos (DLL) al nivel de módulo de formulario, de clase o estándar.

Procedimientos. Un procedimiento **Sub**, **Function** o **Property** contiene partes de código que se pueden ejecutar como una unidad. Se describen en la sección "Introducción a los procedimientos", más adelante en este mismo capítulo.

Módulos de formulario

Los módulos de formulario (extensión de nombre de archivo .frm) son la base de la mayoría de las aplicaciones de Visual Basic. Pueden contener procedimientos que controlen eventos, procedimientos generales y declaraciones a nivel de formulario de variables, constantes, tipos y procedimientos externos. Si examina un módulo de formulario con un editor de textos, podrá ver las descripciones del formulario y sus

controles, así como los valores de sus propiedades. El código que se escribe en un módulo de formulario es específico de la aplicación a la que pertenece el formulario y puede hacer referencia a otros formularios u objetos de la aplicación.

Módulos estándar

Los módulos estándar (extensión de nombre de archivo .bas) son contenedores de los procedimientos y declaraciones a los que tienen acceso otros módulos de la aplicación. Pueden contener declaraciones globales (disponibles para toda la aplicación) o a nivel de módulo de variables, constantes, tipos, procedimientos externos y procedimientos globales. El código que se escribe en un módulo estándar no está ligado necesariamente a una aplicación determinada; si tiene cuidado de no hacer referencia a controles o formularios por su nombre, puede reusar un módulo estándar en distintas aplicaciones.

Módulos de clase

Los módulos de clase (extensión de nombre de archivo .cls) son la base de la programación orientada a objetos en Visual Basic. Puede escribir código en módulos de clase para crear nuevos objetos. Estos objetos nuevos pueden incluir propiedades y métodos personalizados. En realidad, los formularios sólo son módulos de clase que pueden tener controles y que pueden mostrar ventanas de formulario.

Nota Las ediciones Profesional y Empresarial de Visual Basic incluyen también documentos ActiveX, diseñadores de ActiveX y controles de usuario. Estos introducen nuevos tipos de módulos con diferentes extensiones de nombre de archivo. Desde el punto de vista de la escritura de código, estos módulos deben considerarse igual que los módulos de formulario.

1.5 Validación de Entrada

La validación de datos garantiza a la aplicación que cada valor de los datos es correcto y preciso. Puede diseñar la validación de datos en la aplicación con varios enfoques diferentes: código de interfaz de usuario, código de aplicación, restricciones de bases de datos o reglas empresariales.

Éstos son varios tipos de validación de datos.

- Validación del tipo de datos
- Comprobación del rango
- Comprobación del código
- Validación compleja

Una de las formas más sencillas de validación de datos es comprobar los tipos de datos. La validación de tipos de datos responde a preguntas tan simples como "¿Es alfabética la cadena?" y "¿Es numérico el número?" Normalmente podrá manejar tales validaciones con la interfaz de usuario de la aplicación.

Como ampliación del tipo sencillo de validación, la comprobación del rango garantiza que el valor suministrado está dentro de los mínimos y máximos permitidos. Por ejemplo, un código de servicio de tipo de datos de caracteres sólo puede admitir letras alfabéticas de la A a la Z. Los demás caracteres no serán válidos. Como en el caso de la validación de tipos de datos, la interfaz de la aplicación normalmente puede proporcionar la validación de rango necesaria, aunque, como alternativa de diseño, puede crear una regla empresarial para manejar validaciones de rango.

La comprobación del código es un poco más complicada, normalmente requiere una tabla de consulta. Por ejemplo, es posible que la aplicación calcule impuestos sobre ventas sólo para ciertos códigos de estado. Tendrá que crear una tabla de validación para los códigos de estados autorizados sujetos a impuestos. Esta tabla de validación puede ser parte de una regla empresarial o se puede implementar directamente en la base de datos de la consulta.

La validación sencilla de campo y de consulta a veces no es suficiente. Consideremos una petición de asistencia sanitaria que tiene una cantidad facturada de 123,57 unidades monetarias, pero la cantidad permitida puede depender de una acumulación constante del año en curso que se restringe hasta las 1.500 unidades monetarias (para no exceder la política de duración del máximo de 100.000 unidades monetarias). En esta situación, la validez de los datos se amplía más allá de la pantalla de entrada de datos inmediata hasta una cuidadosa evaluación de cómo pagar esta petición basada en los límites de la directiva y las acumulaciones del año en curso. Este tipo de validación de datos compleja se maneja mejor con reglas empresariales.

Una característica desafortunada de antiguas estructuras de archivo es que los datos a menudo se dañan (como campos numéricos que están en blanco o que contienen letras del alfabeto). Cuando genere una aplicación empresarial, deberá generar una utilidad de comprobación para verificar la exactitud de cada campo individual en cada registro de los archivos que utilice dicha aplicación. Si no lo hace así, es posible que la aplicación proporcione resultados impredecibles.

1.6 Compilación de Programas y Generación de Archivos Ejecutables

Hay varias formas de simplificar la compilación y depuración de programas:

Cuando su aplicación no produce resultados correctos, explore el código e intente encontrar las instrucciones que pueden haber causado el problema. Establezca puntos de interrupción en estas instrucciones y reinicie la aplicación.

Cuando el programa se detenga, pruebe los valores de las variables y propiedades importantes. Use Inspección rápida o establezca expresiones de inspección para controlar estos valores. Use la ventana Inmediato para examinar variables y expresiones.

Utilice la opción **Interrupción en todos los errores** para determinar dónde se produjo el error. Para cambiar temporalmente esta opción, seleccione **Alternar** en el menú contextual de la ventana Código y, después, alterne la opción desde el submenú. Recorra paso a paso el código, usando expresiones de inspección y la ventana Locales para controlar cómo cambian los valores mientras ejecuta el código.

Si se produce un error en un bucle, defina una expresión de interrupción para determinar dónde está el problema. Use la ventana Inmediato junto con **Establecer siguiente instrucción** para volver a ejecutar el bucle después de haber hecho las correcciones.

Si determina que una variable o una propiedad está causando problemas en su aplicación, use una instrucción **Debug.Assert** para detener la ejecución cuando se asigna a la variable o a la propiedad el valor incorrecto.

Para establecer el estado de interceptación de errores que Visual Basic utiliza de forma predeterminada antes de empezar cualquier sesión de depuración, abra el cuadro de diálogo **Opciones** (disponible desde el menú **Herramientas**), seleccione la ficha **General** y establezca la opción **Estado de interceptación de errores**. Visual Basic usará este valor la próxima vez que lo inicie, aunque dicho valor se especificara para otro proyecto.

Ocasionalmente, podrá encontrarse un error especialmente difícil de localizar. No se alarme – éstas son algunas cosas que puede hacer:
En primer lugar, haga una copia de seguridad. Este es el punto en el que hasta los programadores más experimentados pierden a menudo muchas horas de trabajo. Mientras se está experimentando, es muy fácil sobreescribir o eliminar accidentalmente secciones de código necesarias.

Use las utilidades de depuración incorporadas a Visual Basic. Trate de identificar la línea o líneas de código que generan el error. Aísle el código. Si puede aislar el problema a un bloque de código, intente reproducir el mismo problema con este bloque de código separado del resto del programa. Seleccione el código, cópielo, inicie un nuevo proyecto, pegue el código en el nuevo proyecto, ejecute el nuevo proyecto y vea si aún se produce el error.

Cree un archivo de registro. Si no puede aislar el código o si el problema es errático, o si el problema sólo ocurre al compilar, entonces la utilidad de depuración de Visual Basic será menos efectiva. En estas situaciones puede crear un archivo de registro que registre la actividad de su programa. Esto le permitirá aislar progresivamente la ubicación del código erróneo. Llame al siguiente procedimiento desde distintos puntos de su programa.

Debe pasar una cadena de texto que indique la ubicación actual del código que se ejecuta en su programa.

```
Sub LogFile (Message As String)
Dim LogFile As Integer
LogFile = FreeFile
Open "C:\VB\LogFile.Log" For Append As #LogFile
Print #LogFile, Message
Close #LogFile
End Sub

Sub Sub1 ()
'...
Call LogFile("Estoy en Sub1")
'...
End Sub
```

Simplifique el problema. Si es posible, quite todos los controles terceros y los controles personalizados de su proyecto. Reemplácelos por controles estándar de Visual Basic. Elimine el código que no parezca tener relación con el problema.

Reduzca el espacio de búsqueda. Si no puede resolver el problema con ninguno de los métodos anteriores, entonces es el momento de eliminar todas las causas ajenas a Visual Basic del espacio de búsqueda del problema. Copie sus archivos AUTOEXEC.BAT y CONFIG.SYS a archivos de copia de seguridad. Quite todos los controladores y programas de estos dos archivos que no sean absolutamente necesarios para ejecutar su programa bajo Windows. Cambie su controlador de vídeo al controlador VGA estándar de Windows. Cierre Windows y reinicie el equipo. Eso eliminará la posibilidad de que haya otro controlador o programa interfiriendo con su programa.

Archivos Ejecutables

La primera etapa en la creación de un programa de instalación personalizado es determinar los archivos que deben distribuirse. Todas las aplicaciones de Visual Basic requieren un conjunto mínimo de archivos, conocido como *archivos de preinstalación*, que son necesarios antes de poder instalar la aplicación. Además, todas las aplicaciones de Visual Basic necesitan archivos específicos de la aplicación como un archivo ejecutable (.exe), archivos de datos, controles ActiveX o archivos .dll.

Existen tres categorías principales de archivos necesarios para ejecutar y distribuir su aplicación:

Archivos de tiempo de ejecución
Archivos de instalación
Archivos específicos de la aplicación

Archivos de ejecución

Los archivos de tiempo de ejecución son archivos que su aplicación debe incluir con el fin de poder funcionar correctamente después de la instalación. Estos archivos son necesarios para todas las aplicaciones de Visual Basic. Los archivos siguientes son los archivos de tiempo de ejecución para los proyectos de Visual Basic:

Msvbvm60.dll
Stdole2.tlb
Oleaut32.dll
Olepro32.dll
Comcat.dll
Asyncfilt.dll
Ctl3d32.dll

Aunque estos archivos son necesarios para todas las aplicaciones de Visual Basic, no siempre lo son para cada uno de los tipos de paquetes de instalación. Por ejemplo, cuando crea un paquete de Internet, el Asistente de empaquetado y distribución supone que cualquier equipo con capacidad para llevar a cabo una descarga a través de Internet dispone ya de todos estos archivos, excepto Msvbvm60.dll. Éste último es, por consiguiente, el único archivo de tiempo de ejecución que el asistente incluye en un paquete de Internet.

Nota Los archivos de tiempo de ejecución pueden clasificarse con más detalle por su ubicación de instalación. Vea "Dónde instalar los archivos en el equipo del usuario" para obtener más información.

Archivos de instalación para paquetes estándar

Los archivos de instalación comprenden todos los archivos necesarios para configurar su aplicación estándar en el equipo del usuario. Incluyen los ejecutables de instalación (setup.exe y setup1.exe), la lista de archivos de instalación (Setup.lst) y el programa de desinstalación (st6unst.exe).

Las aplicaciones de Visual Basic diseñadas para ser distribuidas en disquetes, en CD o en una ubicación de red utilizan los mismos archivos de instalación, sin tener en cuenta si utiliza el Asistente de empaquetado y distribución o el Kit de herramientas de instalación con el fin de crear los programas de instalación. Se enumeran estos archivos a continuación.

Nombre de archivo	Descripción
setup.exe	Programa que el usuario ejecuta con el fin de realizar la instalación previa de los archivos necesarios para la instalación de su aplicación en el equipo del usuario. Por ejemplo, el archivo setup.exe instala el archivo setup1.exe, el archivo DLL de tiempo de ejecución de Visual Basic así como otros archivos sin los que no es posible ejecutar el resto del proceso de instalación.
setup1.exe	Programa de instalación de su aplicación de Visual Basic. El Kit de herramientas de instalación genera este archivo ejecutable y el Asistente de empaquetado y distribución lo incluye en el paquete. Puede volver a nombrar este archivo siempre que el nombre nuevo aparezca reflejado en el archivo Setup.lst.
Setup.lst	Archivo de texto que contiene instrucciones de instalación y que enumera todos los archivos que deben instalarse en el equipo del usuario.
Vb6stkit.dll	Biblioteca con varias funciones utilizadas en Setup1.exe.
St6unst.exe	Herramienta de eliminación de la aplicación.

Nota Las aplicaciones diseñadas para ser entregadas por Internet generalmente no utilizan ninguno de estos archivos. Vea "Paquetes de Internet" previamente en este capítulo para obtener más información acerca de los archivos comprendidos en una entrega de Internet.

Dependencias de la aplicación

Con el fin de ejecutar su aplicación, los usuarios finales necesitarán algunos archivos además de los archivos de tiempo de ejecución comunes y los archivos de instalación especiales. Muchos de estos archivos le parecerán evidentes: el archivo ejecutable, cualquier archivo de datos y cualquier control ActiveX que utilice. Los archivos menos evidentes son los otros archivos dependientes de su proyecto. Por ejemplo, algunos de los controles ActiveX utilizados por su proyecto pueden a su vez necesitar otros archivos. Una de las tareas del Asistente de empaquetado y distribución es determinar la lista completa de los archivos necesarios.

2. EJECUCIÓN Y DEPURACIÓN

2.1 Puntos de Control de la Ejecución

Conforme se escribe un programa en Visual Basic, se pueden tener errores en la sintaxis, esto es, se puede escribir algo como lo siguiente:

Circle (Scalewidth/2, ScaleHeight/2)

Cuando se intenta pasar a la siguiente línea, Visual Basic pone en la pantalla una caja de advertencia indicando que se algo más se está esperando. En este caso se necesita indicar el radio del círculo (por lo menos). De esta manera Visual Basic capta errores a la hora del diseño. Por el otro lado se pueden terminar con errores de Run-Time que son imposibles de evitar en el momento de diseño, tales como fuera de memoria o errores de disco lleno. Una vez que se aprende esto, es posible anticiparse a estos errores, atraparlos y manejarlos.

Los tipos de errores que se discuten a continuación, errores lógicos en el programa, son más difíciles de encontrar. Un error puede estar enterrado en una larga cadena de sentencias complejas. Afortunadamente Visual Basic brinda algunas herramientas de depuración para localizar y hasta corregir errores.

Utilización de Cajas de Texto en Depuración

Las cajas de texto pueden ser excelentes herramientas de depuración, de hecho, son utilizadas para imprimir resultados inmediatos en los programas. Para utilizarlas simplemente hay que agregar unas cajas de texto extras a la aplicación e imprimir valores cruciales conforme va corriendo el programa. Por ejemplo, se desea ver una variable que cuenta los cada vez que se oprime una tecla, también se podría verificar si se están leyendo las coordenadas del ratón correctamente cada vez que se entra a un evento de MouseDown( ). Las cajas de texto temporales brindan una ventana de lo que está pasando atrás de la escena del programa.

Para propósitos de depuración se ordenarán alfabéticamente 10 nombres como los siguientes:

John, Tim, Edward, Samuel, Frank, Tood, George, Ralph, Leonard, Thomas

Se puede hacer el siguiente código que ordene los nombres

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        Names(2) = "Tim"
```

```
        Names(3) = "Edward"
        Names(4) = "Samuel"
        Names(5) = "Frank"
        Names(6) = "Tood"
        Names(7) = "George"
        Names(8) = "Ralph"
        Names(9) = "Leonard"
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Tmp$
                End If
        Next j
Next i
```

Observe que está utilizando el operador lógico > para comparar las cadenas: esto es perfectamente legal en Visual Basic y le permite determinar el orden alfabético de estas cadenas, finalmente se imprime el resultado, nombre por nombre, de la siguiente manera:

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Tmp$
                End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
```

End Sub

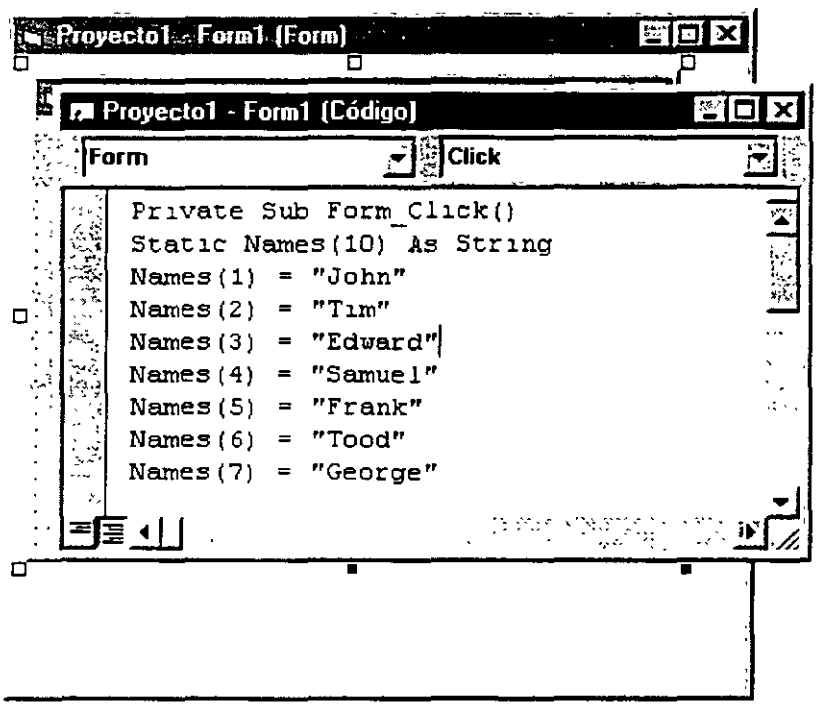
Desafortunadamente el resultado de este programa al correrlo no es satisfactorio:

John

Tim

Todd

Este resultado se ve un poco incompleto, es tiempo de un debug. De hecho un debugging puede ser iniciado sin detener el programa. Para hacer esto, seleccione la opción de Ver Código en el menú de Ver. El procedure de Sub Form_Click( ) aparece como se muestra en la figura siguiente:



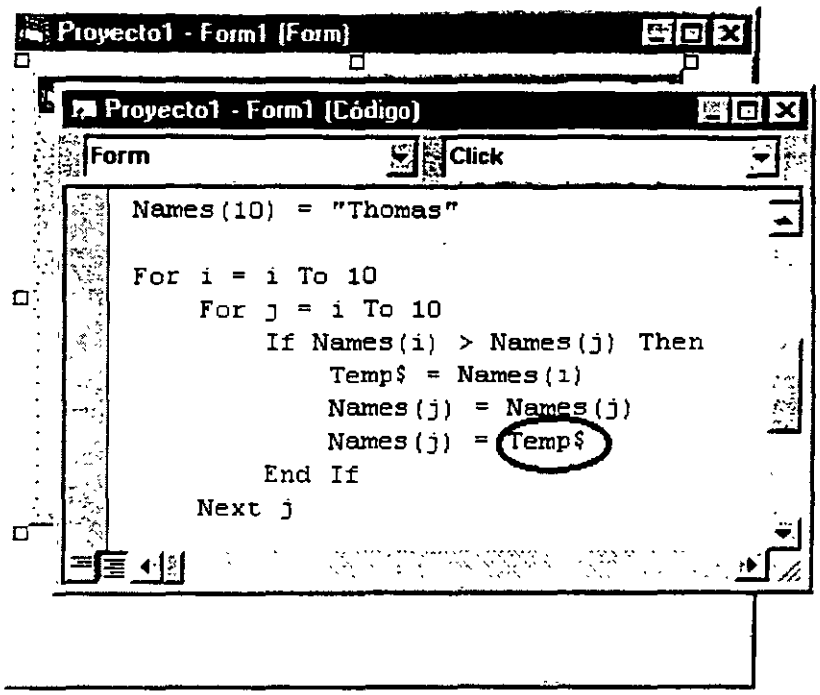
Visual Basic da la posibilidad de revisar el código mientras este está corriendo. Cuando se hace esto se pueden identificar errores inmediatamente con sólo leer el código. En particular cuando se cambian elementos dentro de un arreglo, estos se cargan temporalmente en una variable llamada Temp$; cuando se cargan nuevamente en el arreglo se utiliza una variable que está deletreada incorrectamente como Tmp$:

```
Sub Form_Click ( )
        Static Names(10) As String
```

```
        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Tmp$
                End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
End Sub
```

Los errores de escritura en variables probablemente sea el error lógico más común de Visual Basic. Visual Basic no se queja de estos errores porque asume que se está declarando una nueva variable, tmp$, y la inicializa con valor "".

Este problema puede ser arreglado sin necesidad de terminar el programa. Solamente hay que abrir el menú de Ejecutar de Visual Basic, donde resaltan tres opciones: Interrumpir, Terminar, y Reiniciar. Seleccione Interrumpir para detener el programa temporalmente. Ahora puede cambiar código y continuar con el programa. En la figura siguiente se cambia Tmp$ por Temp$



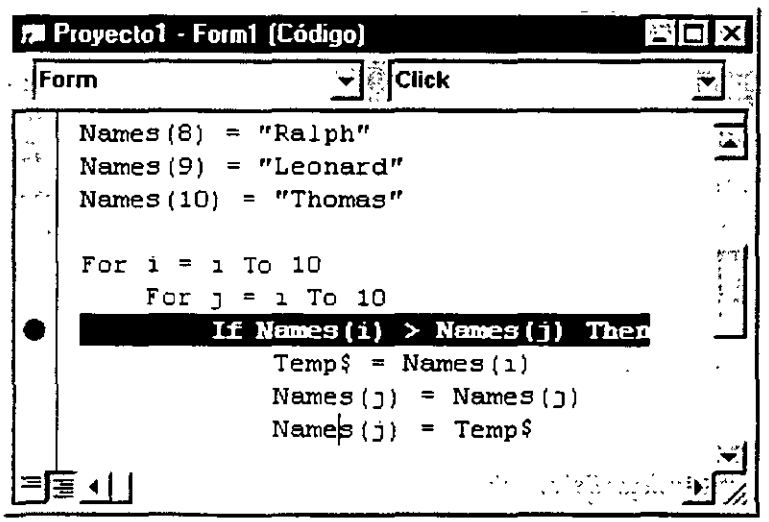
Ahora el programa aparece de la siguiente manera:

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Temp$
                End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
End Sub
```

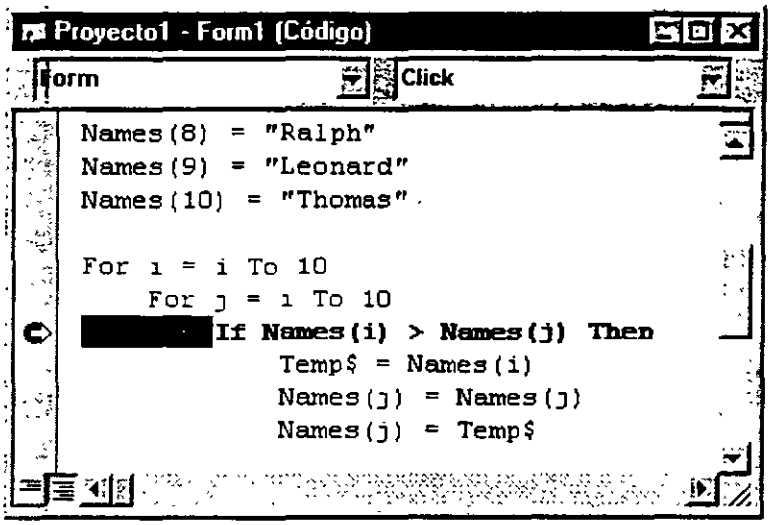
Para que continúe el programa corriendo se puede seleccionar la opción de Continuar en el menú de Ejecutar. Después se puede dar un click a la forma para ver si hubo algún cambio, La lista aparece como la que se muestra a continuación:

John
Tim
Tim
Tim
Tim
Tood
Tood
Tood
Tood
Tood

A pesar de que hubo un cambio el resultado no es aún correcto. El problema obvio en el programa es que las entradas en el arreglo de Names( ) están siendo llenados incorrectamente. Para verificar lo que está sucediendo se deben de observar los elementos en el arreglo conforme se van llenando. Esto se puede hacer estableciendo un punto de interrupción. Esto detiene la ejecución en cierta parte del programa. Para establecer un punto de interrupción se debe de ir a la línea donde se detendrá la ejecución y oprimir F9 o seleccionar Alternar puntos de interrupción en el menú depuración. La sentencia seleccionada aparece en letra negrita de color café para indicar que en ese lugar se estableció un punto de interrupción, como se muestra en la figura.



Después se corre el programa seleccionando Iniciar en el menú de Ejecutar. La ejecución del programa continúa hasta que llega al punto de interrupción. Esta línea se encuentra con bordes y con una flecha como se muestra en la figura.



2.2 Añadiendo Vista de Depuración

Las técnicas de depuración que se presentan en este capítulo usan las herramientas de análisis proporcionadas por Visual Basic. Visual Basic no puede diagnosticar ni solucionar los errores por usted, pero proporciona las herramientas para ayudarle a analizar cómo fluye la ejecución de una parte del procedimiento a otra y cómo cambian los valores de las propiedades y las variables a medida que se ejecutan las instrucciones. Las herramientas de depuración le dejan examinar dentro de la aplicación para ayudarle a determinar lo que pasa y por qué.

La depuración en Visual Basic incluye puntos de interrupción, expresiones de interrupción, expresiones de inspección, paso a paso a través de una instrucción o procedimiento cada vez y presentación de los valores de las variables y las propiedades. Visual Basic también incluye características especiales de depuración, como la posibilidad de modificar y continuar, establecimiento de la próxima instrucción para ejecutar y prueba de procedimientos mientras la aplicación está en modo de interrupción.

Tipos de errores

Para entender de qué manera puede ser útil la depuración, considere los tres tipos de errores que se puede encontrar.

- Errores de compilación
- Errores en tiempo de ejecución
- Errores lógicos
- Errores de compilación

Los *errores de compilación* se producen por un código creado incorrectamente. Si escribe incorrectamente una palabra clave, omite algún signo de puntuación necesario o usa una instrucción **Next** sin la instrucción **For** correspondiente en tiempo de diseño, Visual Basic detectará estos errores cuando compile la aplicación. Los errores de compilación incluyen errores en la sintaxis. Por ejemplo, podría tener una instrucción como sigue:

```
Left
```

Left es una palabra válida en el lenguaje de Visual Basic, pero sin un objeto, no cumple con los requisitos sintácticos para esa palabra (*objeto*.**Left**). Si ha seleccionado la opción **Comprobación automática de sintaxis** en la ficha **Editor** del cuadro de dialogo **Opciones**, Visual Basic presentará un mensaje de error tan pronto como introduzca un error sintáctico en la ventana Código.

Para establecer la opción de Comprobación automática de sintaxis

En el menú **Herramientas**, seleccione **Opciones** y haga clic en la ficha **Editor** del cuadro de diálogo **Opciones**.

Seleccione **Comprobación automática de sintaxis**.

Errores en tiempo de ejecución

Los *errores en tiempo de ejecución* se producen mientras la aplicación está en ejecución (y Visual Basic los detecta) cuando una instrucción intenta una operación que es imposible de realizar. Un ejemplo de esto es una división por cero. Suponga que tiene esta instrucción:

```
Velocidad = Kilómetros / Horas
```

Si la variable **Horas** contiene cero, la división no es una operación válida, incluso aunque la instrucción sea sintácticamente correcta. La aplicación se debe ejecutar antes de que se pueda detectar este error.

Errores lógicos

Los *errores lógicos* se producen cuando una aplicación no actúa de la forma que se pretendía. Una aplicación puede tener código sintácticamente válido, ejecutarse sin llevar a cabo ninguna operación que no sea válida y, aún así, producir resultados incorrectos. Sólo si prueba la aplicación y analiza los resultados puede comprobar que la aplicación está actuando correctamente.

Cómo ayudan las herramientas de depuración

Las herramientas de depuración están diseñadas para ayudarle con:

Los errores lógicos y los que se producen en tiempo de ejecución.

La observación del comportamiento del código que no tiene errores. Por ejemplo, un resultado incorrecto puede producirse al final de una larga serie de cálculos. En la depuración, la tarea es determinar dónde ha fallado algo y qué es lo que ha fallado. Quizá se le ha olvidado inicializar una variable, ha elegido el operador equivocado o ha utilizado una fórmula incorrecta.

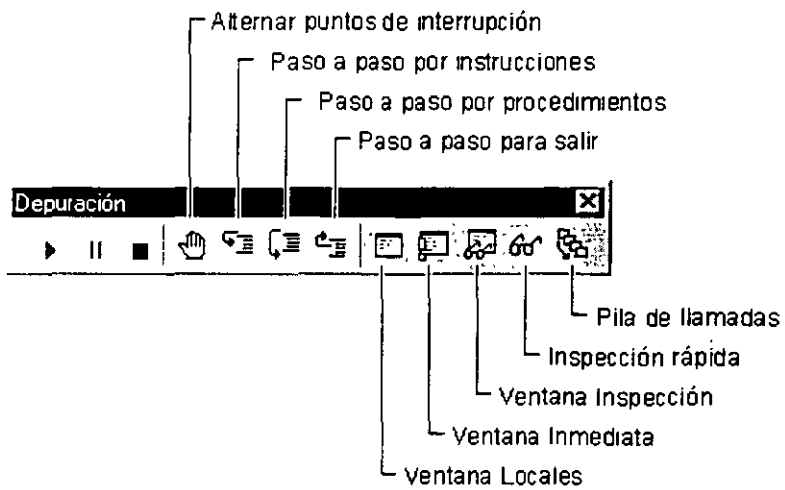
No hay trucos mágicos para depurar y no hay una secuencia fija de pasos que funcione todas las veces. Básicamente, la depuración le ayuda a entender qué es lo que está sucediendo mientras se ejecuta su aplicación. Las herramientas de depuración le ofrecen una instantánea del estado actual de su aplicación, incluyendo lo siguiente:

La apariencia de la interfaz de usuario (UI).
Los valores de las variables, expresiones y propiedades.

Las llamadas a los procedimientos activos.
Cuanto mejor entienda cómo funciona su aplicación, más rápido podrá encontrar los errores.

La barra de herramientas Depurar

Entre sus muchas herramientas de depuración, Visual Basic proporciona varios botones en la barra de herramientas opcional Depurar que son muy útiles. La figura muestra estas herramientas. Para presentar la barra de herramientas Depurar, haga clic con el botón secundario del *mouse* (ratón) en la barra de herramientas de Visual Basic y seleccione la opción **Depurar**.



La siguiente tabla describe brevemente el objetivo de cada herramienta. Los temas de este capítulo tratan las situaciones en las que cada una de estas herramientas le pueden ayudar a depurar o analizar una aplicación con más eficacia.

Herramienta de depuración	Objetivo
Punto de interrupción	Define una línea en la ventana Código donde Visual Basic suspende la ejecución de una aplicación.
Paso a paso por instrucciones	Ejecuta la siguiente línea ejecutable de código de la aplicación y recorre paso a paso las instrucciones de procedimientos.
Paso a paso por procedimientos	Ejecuta la siguiente línea ejecutable de código de la aplicación sin recorrer paso a paso las instrucciones de procedimientos.
Paso a paso para salir	Ejecuta el resto del procedimiento actual y se interrumpe en la siguiente línea del procedimiento de llamada.
Ventana Locales	Presenta el valor actual de las variables locales.
Ventana Inmediato	Le permite ejecutar código o valores de consulta mientras la aplicación está en modo de interrupción.
Ventana Inspección	Presenta los valores de las expresiones seleccionadas.
Inspección rápida	Presenta el valor actual de una expresión mientras la aplicación está en modo de interrupción.
Pila de llamadas	Mientras está en modo de interrupción, presenta un cuadro de diálogo que muestra todos los procedimientos a los que se ha llamado, pero que todavía no se han ejecutado completamente.

2.3 Ejecución Paso a Paso

Si puede identificar la instrucción que causó un error, un único punto de interrupción le puede ayudar a encontrar el problema. No obstante, es más frecuente que sólo sepa el área general del código que ha causado el error. Un punto de interrupción le ayuda a aislar esa área con el problema. Puede usar **Paso a paso por instrucciones** y **Paso a paso por procedimientos** para observar el efecto de cada instrucción. Si es necesario, también puede saltar instrucciones o volver atrás iniciando la ejecución en una línea nueva.

Modo paso a aso	Descripción
Paso a paso por instrucciones	Ejecuta la instrucción actual e interrumpe en la siguiente línea, incluso si está en otro procedimiento,
Paso a paso por procedimientos	Ejecuta el procedimiento entero al que ha llamado la línea actual e interrumpe en la línea siguiente a la línea actual.
Paso a paso para salir	Ejecuta el resto del procedimiento actual e interrumpe en la siguiente instrucción a la que ha llamado al procedimiento.

Nota Para usar estos comandos debe estar en modo de interrupción. No están disponibles en tiempo de ejecución o en tiempo de diseño.

Usar Paso a paso por instrucciones

Puede usar Paso a paso por instrucciones para ejecutar código de instrucción en instrucción. (Esto también se conoce como recorrer paso a paso.) Después de haber recorrido paso a paso cada instrucción, puede ver su efecto si mira en los formularios de la aplicación o en las ventanas de depuración.

Para recorrer paso a paso el código de instrucción en instrucción

En el menú **Depurar**, elija **Paso a paso por instrucciones**.

-o bien-

Haga clic en el botón **Paso a paso por instrucciones** de la barra de herramientas Depurar. (Para ver la barra de herramientas Depurar, haga clic con el botón secundario del *mouse* en la barra de herramientas de Visual Basic y seleccione la opción **Depurar**).

-o bien-

Presione F8

Al usar Paso a paso por instrucciones para recorrer paso a paso el código de instrucción en instrucción, Visual Basic cambia temporalmente a tiempo de ejecución. ejecuta la instrucción actual y avanza a la siguiente instrucción. Luego vuelve a cambiar al modo de interrupción.

Nota Visual Basic le permite recorrer paso a paso instrucciones individuales, incluso aunque estén en la misma línea. Una línea de código puede contener dos o más instrucciones, separadas por un signo de dos puntos (:). Visual Basic usa un contorno rectangular para indicar cuál de las instrucciones va a ejecutar a continuación. Los puntos de interrupción sólo son aplicables a la primera instrucción de una línea que tiene múltiples instrucciones.

Usar Paso a paso por procedimientos

Paso a paso por procedimientos es idéntico a Paso a paso por instrucciones, excepto en que la instrucción actual contiene una llamada a un procedimiento. A diferencia de Paso a paso por instrucciones, que recorre paso a paso por instrucciones el procedimiento al que se ha llamado, Paso a paso por procedimientos lo ejecuta como una unidad y luego recorre paso a paso hasta la siguiente instrucción del

procedimiento actual. Suponga, por ejemplo, que la instrucción llama al procedimiento EstablecerHoraAlarma:

```
EstablecerHoraAlarma 11, 30, 0
```

Si elige **Paso a paso por instrucciones**, la ventana Código mostrará el procedimiento EstablecerHoraAlarma y establecerá la instrucción actual al principio de este procedimiento. Esta es la mejor opción sólo si desea analizar el código dentro de EstablecerHoraAlarma.

Si usa **Paso a paso por procedimientos**, la ventana Código seguirá presentando el procedimiento actual. La ejecución avanza a la instrucción inmediatamente después de la llamada a EstablecerHoraAlarma, a menos que EstablecerHoraAlarma contenga un punto de interrupción o una instrucción **Stop**. Use **Paso a paso por procedimientos** si desea seguir en el mismo nivel de código y no necesita analizar el procedimiento EstablecerHoraAlarma.

Puede alternar con toda libertad entre Paso a paso por instrucciones y Paso a paso por procedimientos. El comando que utilice dependerá las partes de código que desee analizar en un momento dado.

Para usar Paso a paso por procedimientos
En el menú **Depurar**, elija **Paso a paso por procedimientos**.

-o bien-

Haga clic en el botón **Paso a paso por procedimientos** de la barra de herramientas Depurar. (Para ver la barra de herramientas Depurar, haga clic con el botón secundario del *mouse* en la barra de herramientas de Visual Basic y seleccione la opción **Depurar**).

-o bien-

Presione Mayús-F8

Usar Paso a paso para salir

Paso a paso para salir es similar a Paso a paso por instrucciones y a Paso a paso por procedimientos, excepto en que avanza por el resto del código del procedimiento actual. Si al procedimiento lo llamó otro procedimiento, avanza a la instrucción que sigue inmediatamente a la que llamó al procedimiento.

Para usar Paso a paso para salir

En el menú Depurar, elija Paso a paso para salir.

-o bien-

Haga clic en el botón **Paso a paso para salir** de la barra de herramientas Depurar. (Para ver la barra de herramientas Depurar, haga clic con el botón secundario del *mouse* en la barra de herramientas de Visual Basic y seleccione la opción **Depurar**.)

-o bien-

Presione Mayús-F8

Omitir secciones del código

Cuando la aplicación esté en modo de interrupción, puede usar el comando **Ejecutar hasta el cursor para seleccionar una instrucción más abajo en el código** donde quiere que la ejecución se detenga. Esto le permite "saltarse" secciones del código que no son interesantes, como por ejemplo bucles grandes.

Para usar Ejecutar hasta el cursor

1. Ponga su aplicación en modo de interrupción.
2. Coloque el cursor donde desee que se detenga.
3. Presione Ctrl-F8
   -o bien-
   En el menú **Depurar**, elija **Ejecutar hasta el cursor**.

Establecer la siguiente instrucción que se va a ejecutar

Mientras está depurando o experimentando con una aplicación, puede usar el comando Establecer siguiente instrucción para pasar por alto una sección de código determinada, por ejemplo una sección que contenga un error conocido, para así poder rastrear otros problemas. O bien, puede que prefiera volver a una instrucción anterior para probar parte de la aplicación usando valores diferentes para las propiedades o las variables.

Con Visual Basic puede establecer una línea de código diferente para que sea la siguiente que se ejecute, siempre y cuando esté en el mismo procedimiento. El efecto es similar a usar Paso a paso por instrucciones, excepto en que éste sólo ejecuta la siguiente línea de código del procedimiento. Estableciendo la siguiente instrucción que se va a ejecutar, elige la línea que se va a ejecutar a continuación.

Para establecer la siguiente instrucción que se va a ejecutar

1. En modo de interrupción, mueva el punto de inserción (cursor) a la línea de código que desea ejecutar.
2. En el menú Depurar, elija Establecer siguiente instrucción.

3. Para reanudar la ejecución, en el menú Ejecutar, elija Continuar.

   -o bien-

   En el menú Depurar, elija Ejecutar hasta el cursor, Paso a paso por instrucciones, Paso a paso por procedimientos o Paso a paso para salir.

Mostrar la siguiente ejecución que se va a ejecutar

Puede usar Mostrar la siguiente instrucción para colocar el cursor en la siguiente línea que se va a ejecutar. Esta característica es útil si ha estado ejecutando código en un controlador de errores y no sabe con seguridad dónde se va a reanudar. Mostrar la siguiente instrucción sólo está disponible en modo de interrupción.

Para mostrar la siguiente instrucción que se va a ejecutar

1. Cuando esté en modo de interrupción, en el menú Depurar, elija Mostrar la siguiente instrucción.

2. Para reanudar la ejecución, en el menú Ejecutar, elija Continuar.
   -o bien-

   En el menú Depurar, elija Ejecutar hasta el cursor, Paso a paso por instrucciones, Paso a paso por procedimientos o Paso a paso para salir.

3. CONTROLES Y OBJETOS PARA EL MANEJO DE DATOS

3.1 Características de los objetos relacionados a datos

En la terminología orientada a objetos de OLE 2.0, los objetos son contienen propiedades, métodos y otros objetos. Los métodos son llamados funciones miembros de un objeto: estos ejecutan una acción dentro del objeto como cambiar el color, tamaño o forma de un objeto. Las propiedades son funciones miembros pares de un objeto programable; se puede establecer o regresar información acerca del estado de un objeto programable , tal como el valor de un elemento de datos o un campo dentro de una tabla. Una función miembro establece los datos y otra función regresa los datos, debido a esto se denominan pares. Se puede decir que todas las funciones miembros están encapsuladas en el objeto.

Visual Basic le da acceso a todas las funciones miembro de sus objetos programables nativos, tales como los objetos de forma y acceso de datos. Las aplicaciones de automatización de OLE son selectivas en cuanto a los objetos programables y las funciones miembro que son accesibles para otras aplicaciones. El hacer funciones miembros de las aplicaciones fuente de automatización de OLE accesibles a las aplicaciones contenedoras de automatización de OLE se conoce como exponer la función miembro.

El objeto de data access en Visual Basic incluye a todos los objetos que le permiten conectarse y manipular los diferentes tipos de bases de datos que se soportan. El término objeto compuesto se usa para describir a un objeto que contiene a otros objetos para mantener consistencia con la terminología de OLE de documento compuesto. Al igual que los documentos compuestos, los objetos compuestos tienen una estructura jerárquica. Los objetos que se encuentran dentro de otros objetos se les conoce como objetos miembros del objeto contenedor. Visual Basic trata a los objetos miembros como propiedades del objeto contenedor. La figura 3.1 muestra la jerarquía de los objetos de base de datos en Visual Basic.

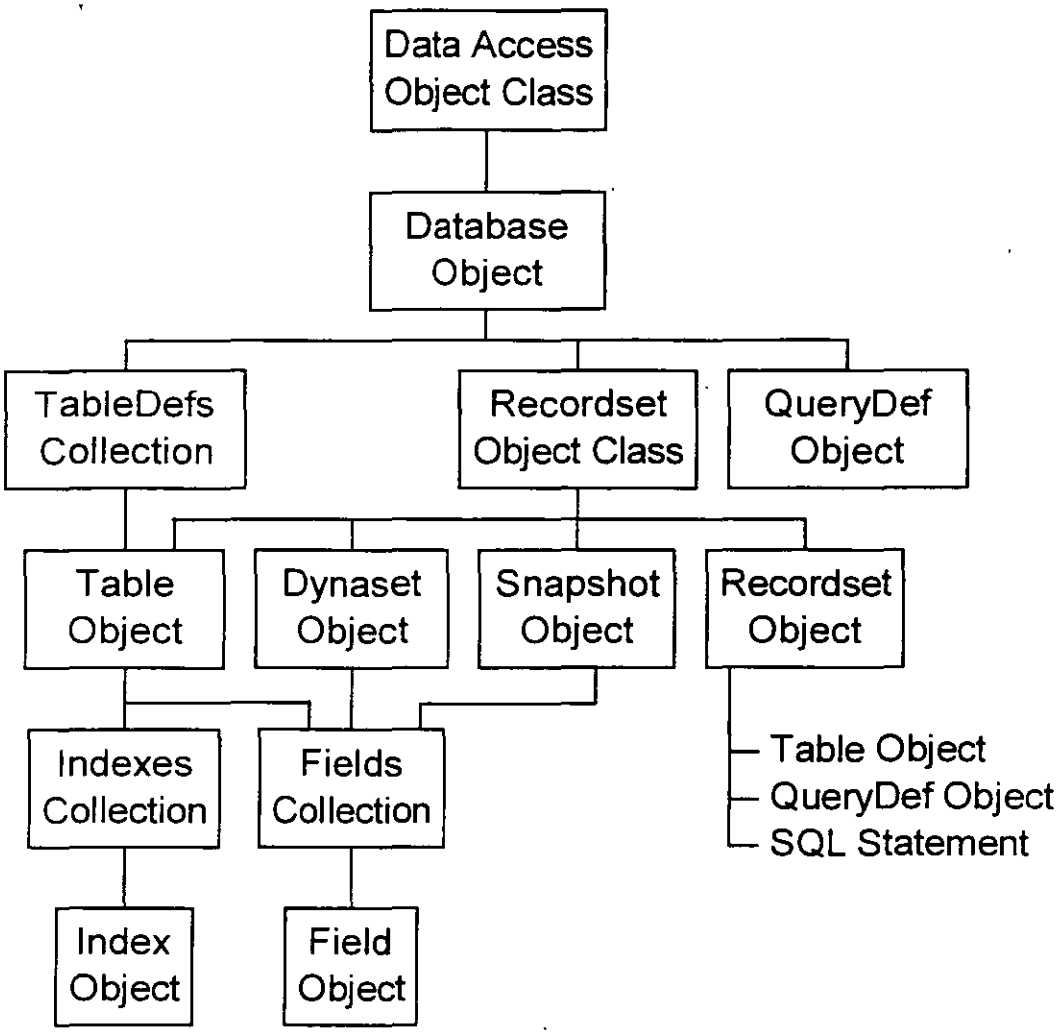


fig. 3.1

Principios de los objetos de Data Access

Se puede crear una instancia de un objeto de acceso de datos cuando se agrega un objeto de data control a una forma de Visual Basic. Una instancia de un objeto es una referencia con nombre que apunta al objeto, no es el objeto en sí . Con el objeto de data access el objeto miembro Database se convierte en una propiedad del data control. Se direcciona al objeto de Database de un data control con sentencias que se refieran a DataControlName.Database.PropertyName. Solamente el objeto de data control crea una instancia para el objeto de data access en si, a esto se le conoce como DataControlName en la sentencia anterior.

Usted no puede crear una instancia del objeto de data access con código de Visual Basic, debido a que no existen objetos de tipo DataAccess en el lenguaje de Visual Basic. Usted

puede crear una instancia del objeto de tipo Database (Object class) en un proceso de dos pasos que se describe a continuación:

1. Declarar una variable de objeto con la sentencia

Dim dbname **As Database**

2. Crear un puntero al nuevo objeto de base de datos con la sentencia

Set dbname = **OpenDatabase**(ConnectArguments)

Es posible crear múltiples instancias del mismo objeto miembro. Cada instancia del objeto miembro es independiente de las otras instancias dentro del objeto. Por ejemplo: es posible crear múltiples instancias en el mismo objeto de base de datos en una sola aplicación al agregar un objeto de data control de una sola base de datos a más de una forma. El objeto Database de cada data access es idéntico para cada instancia, pero las propiedades de los objetos miembros de cada instancia del objeto Database, tal como la propiedad Recordset del data control son independientes para otras instancias.

La palabra **New** de Visual Basic le permite crear una nueva instancia de un objeto que ya existe o ya fue definido sin tener que utilizar la sentencia **Set** cada vez que usted desea crear una instancia para un objeto. Visual Basic crea una nueva instancia de un objeto declarado con la palabra **New** la primera vez que se refiera a un método o propiedad de una nueva instancia. La sintaxis general de las sentencias **Dim** que usan la palabra **New** es:

Dim objname **As New** ***ObjectType***

Objetos Miembros Persistentes

Los objetos miembros persistentes son objetos cuyas propiedades se encuentran en un archivo. Los objetos persistentes son llamados comúnmente objetos físicos; las propiedades de los objetos persistentes son independientes de la aplicación. Los siguientes miembros de objeto de base de datos son persistentes:

- Los objetos de **Table** y **Field** son objetos miembros persistentes de todos los tipos de bases de datos. Una tabla debe de tener al menos un campo para ser considerada así. Todos los métodos de la clase de objetos **Recordset** aplican a los objetos de tabla, exceptuando a los métodos **Clone** y **Find**.

- Los objetos **QueryDef** son objetos persistentes que solo están disponibles cuando se conecta a una base de datos Access. Los **QueryDef** que contienen una sentencia de SQL son almacenados en el archivo .MDB de Access. Los objetos **QueryDef** no contienen datos sobre las tablas involucradas en un query.

- Los objetos **TableDef** e **Index** son persistentes para Access, Btrieve y para la mayoría de bases de datos Cliente-Servidor, debido a que las tablas y sus definiciones son almacenadas en una o más tablas dentro de la base de datos. Si

la tabla no está indexada, no existirán objetos **Index** para el objeto **Table** especificado.

Los cambios que se hagan en la aplicación a los objetos miembros persistentes aparecen en cada instancia del objeto persistent data en la aplicación al igual que las instancias del mismo objeto miembro en otras aplicaciones que corren dentro de un ambiente multiusuario. Sin embargo los cambios hechos a objetos persistentes en otras aplicaciones no aparecerán hasta que el objeto sea reabierto o actualizado.

Objetos creados de tablas virtuales

Una tabla virtual es una copia o imagen de in objeto **Table** o el conjunto de columnas y renglones que regresa un query de SQL que esta corriendo en base a una o más tablas. Las tablas virtuales son almacenadas en RAM y no tienen ninguna manifestación física: usted no puede copiar una tabla virtual a un archivo de disco. Si la imagen es más grande que la cantidad de memoria RAM disponible para la aplicación, porciones de la tabla virtual son paginadas a un archivo temporal localizado en el directorio especificado por la entrada SET TEMP= en el archivo AUTOEXEC.BAT. La tabla virtual existe solo mientras dure la vida de la variable del objeto data type que apunta a la tabla virtual. Se puede escoger entre crear una tabla virtual o una tabla estática.

Tablas virtuales dinámicas

Las tablas virtuales dinámicas, al igual que los objetos miembros persistentes, reflejan cambios hechos por otros en un ambiente multiusuario a los objetos persistentes de la base de datos. La aplicación ve la versión más actual de las tablas físicas que caen dentro de los objetos miembro. La aplicación puede alterar los valores de la mayoría de las tablas virtuales dinámicas más no de todas. Los objetos miembro que están basados en tablas virtuales son:

- objetos **Dynaset** son tablas virtuales creadas a partir de un objeto **QueryDef** o uno o más objetos **Table**. Todos los métodos que aplican a los objetos tipo **Recordset** con excepción del método **Seek,** aplican a los objetos **Dynaset**. Los objetos **Dynaset** no cuentan con la propiedad de **Index**. Puede utilizarse la propiedad de Sort de un **Dynaset** para crear un **Dynaset** que este ordenado bajo un campo del **Dynaset** original.

- El objeto **Recordset** son objetos **Dynaset** que tienen tres métodos adicionales (Refresh, UpdateControls y UpdateRecord) que no aplican al **Dynaset** en sí.

- Los objetos **TableDef** e **Index** para las tablas de Dbase, FoxPro y Paradox son creadas bajo un proceso un poco diferente. Aquí se crea una tabla virtual **TableDef** mediante un análisis del encabezado del archivo .DBF o .DB. De manera similar se cera la colección de índices y sus miembros al leer los encabezados de los archivos .MDX, .NDX y .PX.

Tablas virtuales estáticas

El objeto **Snapshot** es el miembro estático del objeto **Database**. Un objeto **Snapshot** captura una imagen estática de un objeto miembro persistente o un objeto **Dynaset**. Los valores que contiene los objetos **Snapshot** son de sólo lectura todo el tiempo. Se pueden aplicar cualquiera de los métodos del objeto **Dynaset** a objetos convencionales **Snapshot.**

El método **ListTable ( )** del objeto **Database** y los métodos de **ListFields ( )** y **ListIndexes ( )** para los objetos de **Table** crean un tipo especial de objeto **Snapshot** al cual no se le puede aplicar el método **Find...**

3.2 El objeto Database

Antes de obtener o establecer las propiedades de un objeto **Database** o aplicar métodos a un objeto **Database**, es necesario crear una variable con nombre del objeto **Database** data type o agregar un objeto de data control a una forma. Antes de esto ya se debió de haber creado una referencia o apuntador al objeto de **Database** con las siguientes sentencias:

Dim dbname **As Database**

Set dbname = **OpenDatabase**(ConnectArguments)

Técnicamente **OpenDatabase( ), CreateDatabase( )** y **SetDefaultWorkspace( )** deben de ser clasificados como métodos del objeto de data access de Visual Basic, debido a que no se puede aplicar un método a un objeto para el cual no existe una referencia. Clasificando estas funciones como métodos del objeto de data access es consistente con la clasificación del objeto de data control como una instancia del objeto de data access. Sin embargo para consistencia con la documentación de Visual Basic, estos métodos son clasificados como métodos del objeto **Database.**

Propiedades del objeto Database

El objeto **Database** tiene cinco propiedades cuyos valores pueden ser leídos para determinar las características de la base de datos como un todo. La mayoría de estas propiedades son de sólo lectura todo el tiempo. La siguiente tabla lista las propiedades del objeto **Database**, en un orden aproximado de frecuencia en el que pueden ser utilizadas estas propiedades.

Propiedad	Propósito
Name	Regresa una cadena que contiene la ruta y el nombre de la base de datos abierta. El valor de esta propiedad del objeto de data control puede leerse o establecerse.

Propiedad	Propósito
Connect	Regresa el valor de la cadena utilizada para establecer una conexión a la base de datos. La propiedad de connect para el objeto **Database** es de sólo lectura y está vacía para las bases de datos de Access. La propiedad connect de un objeto data control puede leerse o asignarse.
Updatable	Indica si la base de datos ha sido abierta en modo lectura-escritura (True) o modo sólo lectura (False). Esta propiedad en sí es de sólo lectura.
Transactions	Indica si la tabla soporta la sentencia **Rollback** que le permite deshacer un grupo de cambios a los valores de los datos en las tablas de la base de datos. Access y la mayoría de las bases de datos Cliente-Servidor soportan esta función.
QueryTimeout	Especifica o indica la cantidad de tiempo en segundos antes de que ocurra un error de time-out cuando se ejecuta un query para una base de datos Cliente-Servidor en un driver ODBC. El valor por default es de 60 segundos.
CollatingOrder	Es una bandera Integer que indica el lenguaje cuyas reglas se están utilizando para ordenar campos de texto. El valor por default es 256 (Inglés y la mayoría de los lenguajes del oeste europeo). Esta propiedad es se sólo lectura a excepción de cuando se utiliza los métodos de **CreateDatabase( )** y **CompactDatabase( )**.

El valor de la propiedad CollatingOrder puede ser utilizado como el valor opcional del argumento int*Compare* de las funciones **InStr( )** y **StrComp( )** como se muestra en el siguiente fragmento de código. Si se establece el valor de int*Compare* al valor de la propiedad CollatingOrder , a cualquier valor excepto -1, ignora el método de comparación por default especificado por la sentencia **Option Compare** en la sección de declaraciones del módulo.

```
Dim initcompare As Integer
Dim initInStr As Integer
Dim varStrComp As Variant

intCompare = dbaname.CollatingOrder
intInStr = InStr(StrSource. strTest. intCompare)
VarStrComp = StrComp (str1, str2, intCompare)
```

Métodos Aplicables al objeto Database

El objeto **Database** tiene muchos más objetos que propiedades. El método de **Opentable()** es tratado como un método del objeto **Database** para consistencia con la clasificación del método **OpenQueryDef( )**. El método **OpenQueryDef( )** es un método del objeto **Database** debido a que los objetos de tipo **QueryDef** pueden actuar en más de una tabla del objeto **Database**. Los métodos que pueden aplicarse a un objeto **Database** se presentan en la siguiente tabla. Estos métodos están agrupados en grupos de métodos relacionados en lugar de orden alfabético.

Método	Propósito
OpenDatabase( )	Utilizado en la sentencia de **Set** para crear una referencia nombrada a una base de datos existente. Se puede usar este método para crear nuevas bases de datos de escritorio en lugar de bases de datos de Access.
OpenTable( )	Utilizado en la sentencia de **Set** para crear un objeto de tabla con nombre que represente una tabla existente contenida en un objeto de **Database** creado previamente.
Close	Cierra un objeto de **Database** y libera recursos consumidos por el objeto. Todos los objetos miembros del objeto de **Database** deben de ser cerrados antes de cerrar este objeto. Si ha declarado la variable del objeto **Database** con alcance local, no es necesario utilizar este método debido a que el objeto **Database** y sus miembros se cierran cuando las variables salen del alcance al momento en el que termina el procedimiento donde las variables fueron declaradas.
CreateDynaset( )	Utilizado en la sentencia de **Set** para crear un **Dynaset**, una tabla virtual dinámica.
CreateSnapshot( )	Utilizado en la sentencia de **Set** para crear un **Snapshot**, una tabla virtual estática
OpenQueryDef( )	Utilizado en la sentencia de **Set** para crear un objeto **QueryDef** con nombre que representa un **QueryDef** existente en una base de datos Access. Los objetos **QueryDef** están disponibles para bases de datos Access únicamente.

Método	Propósito
CreateQueryDef( )	Utilizado en la sentencia de **Set** para crear un nuevo objeto persistente **QueryDef** basado en una sentencia de SQL.
DeleteQueryDef( )	Borra un elemento persistente **QueryDef** de una base de datos Access. Si el elemento **QueryDef** no existe se incurre en un error de runtime.
BeginTrans( )	Indica en inicio de una serie de operaciones relacionadas que actualizan valores en uno o mas objetos **Recordset**
CommitTrans( )	Especifica el final de una actualización a un objeto **Recordset** que constituye una sola transacción y hace que los cambios se apliquen sobre los objetos **Table** incluidos en los **Recordset**
Rollback	Si el tipo de base de datos soporta transacciones. cancela la actualización de los objetos **Table** incluidos en los **Recordsets.**
ListTable( )	Utilizado en la sentencia de **Set** para crear un objeto **Snapshot** que contenga definiciones de cada una de las tablas en una base de datos. Las definiciones son idénticas a los miembros TableDef de la colección TableDefs.
SetDataAccessOption	Establece una constante variante global subtipo 8 que tiene una ruta bien definida y el nombre del archivo APPNAME.INI para la aplicación en caso de que este archivo no se encuentre dentro del directorio de windows.
SetDefaultWorkspace	Establece un conjunto permanente de valores de UID (user ID) y PWD (password) que aplican a todas las bases de datos Access aseguradas.
ResgisterDatabase( )	Crea una sección o actualiza una sección existente del archivo \WINDOWS\ODBC.INI que se usa para registrar entradas que contienen datos fuentes ODBC para todas las aplicaciones que usan fuentes de datos ODBC.
CreateDatabase( )	Crea una base de datos Access con el nombre u la

Método	Propósito
	localización especificadas en el primer argumento. El segundo argumento es opcional y especifica la propiedad de CollatingOrder y el tercer argumento (opcional) especifica si la base de datos se va a encriptar y si la nueva base de datos se crea en formato Access 1.0 o 1.1.
CompactDatabase	Compacta una base de datos de Access para recuperar espacio utilizado por registros borrados y elementos QueryDef.
RepairDatabase	Intenta reparar una base de datos Access que fue dañada por un evento anormal, tal como una falla en el suministro de corriente durante una actualización.
Freelocks	Suspende temporalmente la ejecución de la aplicación para permitir actualizaciones a los objetos **Database** en primer plano y para actualizar los datos contenidos en los objetos **Recordset** de la aplicación. **Freelocks** está relacionado a la sentencia de **DoEvents** que temporalmente suspende la ejecución de la aplicación para permitir que windows procese mensajes en el queue de mensajes.

3.3 El acceso a las bases de datos utilizando Visual Basic

Existen tres tipos de bases de datos que se pueden accesar desde Visual Basic (fig 3.2):

- Bases nativas de Microsoft Access. Estas bases de datos son accesadas directamente por Visual Basic.
- Bases de datos indexadas con método de acceso secuencial (ISAM). Por ejemplo bases de datos de Dbase. Paradox y Btrieve. Visual Basic obtiene estas tablas a través de drivers instalados por el usuario que ligan a Visual Basic a bases de datos específicas.
- Bases de datos en sistemas abiertos (ODBC). Estas incluyen sistemas de administración de bases de datos Cliente-Servidor tales como Microsoft SQL Server y Oracle. Visual Basic obtiene estas bases a través de los drivers ODBC correspondientes.

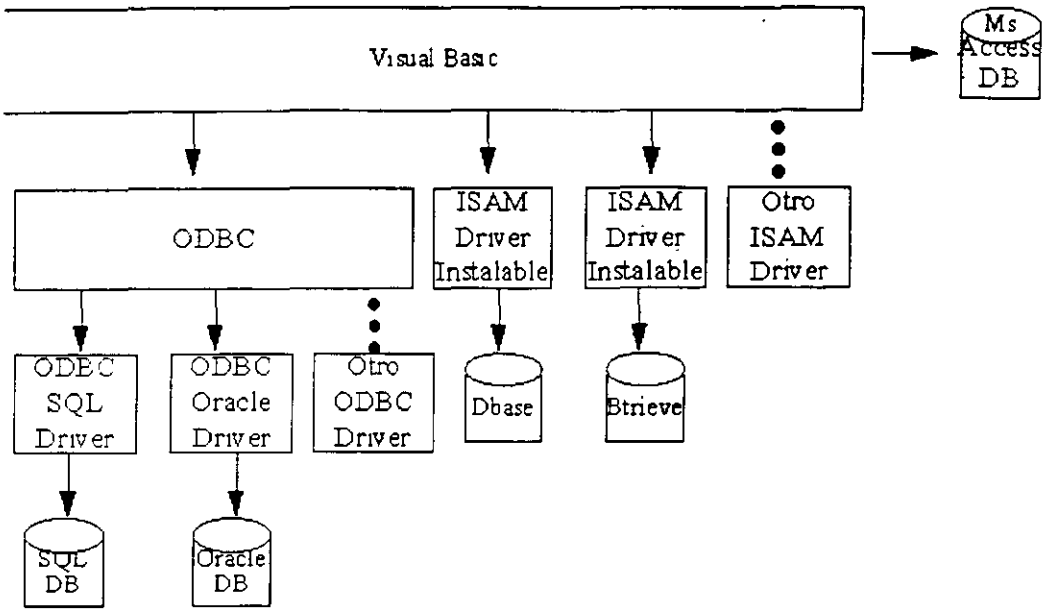


fig. 3.2

Si la aplicación espera abrir una base de datos Access que ha sido asegurada que tiene definida un esquema de permisos, es necesario añadir una línea en el archivo VB.INI o en el archivo APP.INI para indicar la ubicación del archivo .MDA, por ejemplo:

[options]

SystemDB=C:\ACCESS\SYSTEM.MDA

Además la aplicación debe de incluir la sentencia de **SetDefaultWorkspace** para indicar un nombre de usuario y un password apropiado. Esta sentencia debe de ir antes de cualquier otra sentencia que haga empezar a la inicialización del motor de bases de datos de Visual Basic .

SetDefaultWorkspace "myusername", "mypassword"

3.4 El objeto Data Control, sus métodos y propiedades

El objeto data control automatiza el proceso de abrir una base de datos y crear un objeto **Recordset.** El objeto **Recordset** es una tabla virtual que contiene un conjunto de elementos de datos necesarios para la aplicación. La forma más simple de un objeto **Recordset** es la imagen de una tabla, similar al VIEW creado con la sentencia de SQL CREATE VIEW *nombre* AS SELECT * FROM *tabla*. La diferencia entre un objeto **Recordset** y una vista convencional es que los datos en el objeto **Recordset** son actualizables si se tiene acceso de lectura-escritura a la base de datos y a la tabla; las vistas creadas con SQL son de sólo lectura.

Propiedades del objeto data control

La siguiente tabla muestra el nombre, tipo de acceso, descripción y uso de cada una de las propiedades relacionadas a datos del objeto data control. El resto de las propiedades de data control tales como: Caption, Enabled y Visible, se comparten con la mayoría de los otros objetos de control de Visual Basic. Las propiedades están enlistadas en el orden que aparecen en la ventana de propiedades del objeto data control

Nombre	Modo de Diseño	Modo de Corrida	Descripción y uso
Database	NA	RO	Un apuntador al objeto **Database** que cae sobre el data control. El valor de la propiedad **Database** se utiliza para direccionar un objeto **Database** al data control en el código de Visual Basic.
DatabaseName	RW	RW	La ruta bien formada hacia el directorio que contiene las bases de datos de escritorio además del nombre del archivo y la extensión .MDB de una base de datos de Access o bien el archivo de definición de datos FILES.DDF de una base de datos de Btrieve. La propiedad de DatabaseName se deja vacía cuando se especifica el nombre de la fuente de datos en la propiedad Connect, o si desea que aparezca la caja de diálogo de ODBC. Es posible utilizar la sintaxis de UNC para sustituir la ruta a archivos que están localizados en servidores que soportan nombres uniformes convencionales.
Connect	RW	RW	Se utiliza para determinar el tipo de base de datos. Esta propiedad se deja vacía cuando se trata de una base de datos Access o si se desea que aparezca la caja de diálogo de ODBC. Para conectarse a archivos de Dbase, Paradox, FoxPro o Btrieve se debe de digitar el nombre de conexión estándar seguido de dos puntos (:) **Paradox:, Dbase III:, Dbase IV:, FoxPro 2.5:,** o **Btrieve:**. Para conectarse a una base de datos con ODBC API, digitar **DSN=Nombre del Data Source;**

Nombre	Modo de Diseño	Modo de Corrida	Descripción y uso
			UID=UserID, PWD=Password.
RecordSource	RW	RW	Especifica la fuente de datos sobre el objeto **Recordset**. La propiedad RecordSource puede ser el nombre de la tabla de la base de datos, un elemento QueryDef de una base de datos Access o una sentencia válida de SQL. Si se pone la bandera de la propiedad de Opciones DB_SQLPASSTHROUGH, se pueden mandar sentencias de ANSI SQL a bases de datos conectadas con el ODBC API
Recordset	NA	RO	Un objeto **Dynaset** definido por las propiedades Databasename, Connect y Recordsource del objeto de control de datos. Si se utiliza la edición profesional de Visual Basic, todas las propiedades y métodos del objeto **Dynaset** aplican al objeto **Recordset.**
Options	RW	RW	Es una bandera larga que establece y regresa los valores que determinan el acceso de la aplicación y el acceso a otras aplicaciones a las tablas que están incluidas en el objeto **Recordset**. El valor por default es 0.
Exclusive	RW	RW	Una bandera entera que determina el candado (locking) de la base de datos especificado por Databasename o Connect. Esta tiene valor de True cuando se utiliza un acceso exclusivo y niega el acceso a otros para abrir la base de datos mientras su aplicación tenga la base de datos abierta. El valor False está dado por default para acceso compartido.
ReadOnly	RW	RW	Una bandera entera que determina si la aplicación puede actualizar valores de

Nombre	Modo de Diseño	Modo de Corrida	Descripción y uso
			datos de las tablas en una base de datos abierta. El valor True es para acceso de sólo lectura y el valor por default. False. es para acceso lectura-escritura.

La siguiente tabla especifica los valores de la bandera de Options utilizada para controlar el acceso a tablas que participan en la creación del objeto **Recordset** del objeto de data control. Digite el valor decimal de la bandera en la caja de texto de Options de la ventana de propiedades para un control de data. Se pueden utilizar las constantes simbólicas globales mostradas en la tabla a continuación para hacer más leíble el código de Visual Basic. Para utilizar las constantes simbólicas cargue el archivo de DATACONS.TXT en la sección de declaraciones del módulo de la aplicación

Constante	Valor	Propósito
DB_DENYWRITE	1 (&H1)	Evita que otras aplicaciones hagan cambios a los valores de datos contenidos en las tablas que participan en abrir el objeto **Recordset** de la aplicación.
DB_READONLY	4 (&H2)	Crea un **Recordset** que no es actualizable.
DB_APPENDONLY	8 (&H8)	Permite a la aplicación agregar un registro nuevo al **Recordset** pero no lee ni actualiza los registros existentes.
DB_INCONSISTENTUPDATES	16 (&H10)	Permite modificar un valor de un campo de un objeto **Recordset** que provoca una serie de cambios a los valores de otros campos del **Recordset.** Le permite hacer cambios a un lado de las

Constante	Valor	Propósito
		tablas unidas en una relación de uno a muchos.
DB_CONSISTENTUPDATES	32 (&H20)	No le permite modificar un valor de un campo de un objeto **Recordset** que provoque una serie de cambios a los valores de otros campos del **Recordset.**
DB_SQLPASSTHROUGH	61 (&H30)	Permite enviar sentencias de SQL directamente al administrador de driver de ODBC y del administrador de driver a cualquier base de datos que se conecte con ODCB API. Un RDBMS de Cliente-Servidor procesa la sentencia de SQL en el servidor y regresa los registros que especificó la sentencia de SQL.

Métodos del objeto data control

Los métodos que aplican a los objetos **Dynaset** también aplican a los objetos **Recordset.** Sin embargo el objeto data control tiene dos métodos exclusivos y utiliza el método de Refresh en una manera muy especial. La siguiente tabla muestra los tres métodos relacionados a datos particulares del objeto Data Control.

Método	Propósito
UpdateRecord	Guarda los valores de todos los controles de datos ligados a los campos correspondientes de las tablas en la base de datos. Aplicar este método tiene un efecto similar al método **Edit**. Estos métodos cambian el foco a otro control ligado en otro campo y después aplican el método **Update**. Sin embargo no se generan eventos **Validate ( )** durante el proceso. Es muy común invocar este método cuando el usuario da un click en OK. Save o Update.
UpdateControls	Es la contraparte del método UpdateRecord. Este método

Método	Propósito
	reemplaza valores en controles ligados con los valores del campo del registro actual. Este método se invoca por lo general cuando se da un click en Cancel o Undo.
Refresh	Abre una nueva base de datos o vuelve a abrir una ya existente y recrea el **Dynaset** que sirve como el control de datos del objeto **Recordset.**

Las sentencias que contengan los métodos UpdateRecord y Update Controls son permitidas solamente dentro de procedimientos de manejo de eventos con Validate( ). Cuando el control es instanciado en conjunción con la apertura del objeto forma que lo contiene el método Refresh se aplica automáticamente al data control. Después de cambiar el valor de cualquiera de las propiedades del data control se necesita aplicar el método de Refresh. Este método crea la tabla virtual del objeto **Recordset.** Así que cuando la aplicación esté diseñada para ambiente multiusuario, puede utilizarse el método Refresh para asegurarse que los valores de datos en el en el data control del **Recordset** reflejen los valores actuales de los datos en las tablas.

4. MANEJO DE BASES DE DATOS DE ACCESS

4.1 Arquitectura de las Bases de Datos de Access

La mayoría de las aplicaciones complejas de Windows usan una estructura de capas comprimida en un archivo ejecutable (.EXE) y una o más librerías dinámicas ligadas (DLL's) que la aplicación llama cuando las necesita.

Los archivos ejecutables de Visual Basic que usted "compila" contienen el equivalente de código objeto de Visual Basic para las formas y código fuente de su aplicación. La principal diferencia entre librerías DOS y Windows es que el código objeto en librerías para las aplicaciones DOS se convierte en una parte permanente del archivo .EXE (llamadas ligas estáticas). mientras que las DLLs de Windows son ligadas en tiempo de ejecución (llamadas librerías dinámicas). Una de las ventajas de las ligas dinámicas es que una simple .DLL. tal como VBRUN300.DLL. puede servir a todas las aplicaciones de Visual Basic en su computadora. Si usted ejecuta mas de un aplicación de Visual Basic al mismo tiempo. Windows crea instancias adicionales de VBRUN300.DLL en memoria para servir a las otras aplicaciones.

La figura 4.1 ilustra la estructura de capas de una aplicación de base de datos de Visual Basic. La siguiente lista explica la lista de eventos que ocurren cuando su aplicacion llama a una operación de acceso de datos .usando la terminología de la figura 4.1:

- El archivo .EXE de su aplicación envia un requerimiento a VBRUN300.DLL para realizar una operación en un objeto de acceso a datos.

- VBRUN300.DLL pasa el requerimiento de acceso a datos al DLL de Soporte de Acceso a Datos. VBDB300.DLL.

- VBDB300.DLL traduce el requerimiento a llamadas a función que son compatibles con las llamadas a funciones de la librería JET Engine. MSAJT110.DLL. ésta determina el tipo de base de datos que corresponde al objeto. abre una instancia del controlador de base de datos apropiado y pasa la llamada al controlador.El controlador devuelve el dato (suponiendo que la petición fue para alimentar valores contenidos en los campos de la tabla) respaldando la cadena de DLLs a su aplicación.

- Si su aplicación utiliza en ODBC API. otra capa (el ODBC.DLL) es interpuesto entre el DLL JET Engine y el controlador ODBC que se conecta a la base de datos.

Tres de los elementos mostrados en la figura 4.1 son opcionales para las aplicaciones de bases de datos de Visual Basic. ésos son descritos en la siguiente lista:

- La aplicación ODBC Administrator provista con Visual Basic viene en dos versiones: una aplicación stand-alone. ODBCADM.EXE y ODBCINST.DLL. la que está diseñada para ser añadida al panel de control.

- Si su base de datos de dBASE o FoxPro incluye archivos de indices. necesitará un archivo .INF en su directorio de la base de datos que especifique el índice para cada tabla.

- Access 1.x requiere un archivo SYSTEM.MDA que contiene información acerca de los grupos de usuarios de las bases de datos de Access. Siusted aplica cualquiera de las opciones de seguridad de Access 1.x a un archivo de base de datos de Access. un archivo SYSTEM.MDA sencillose requiere para el mecanismo de base de datos de Access para abrir el archivo.Si usted no ha establecido seguridad en el acceso a las bases de datos con Access 1.x. no necesitará SYSTEM.MDA. En este caso. todos los usuarios son identificados como Admin (un miembro del grupo Admins) y tienen un password (Null).

La siguiente figura nos muestra la estructura de los manejadores de bases de datos respecto a Visual Basic:

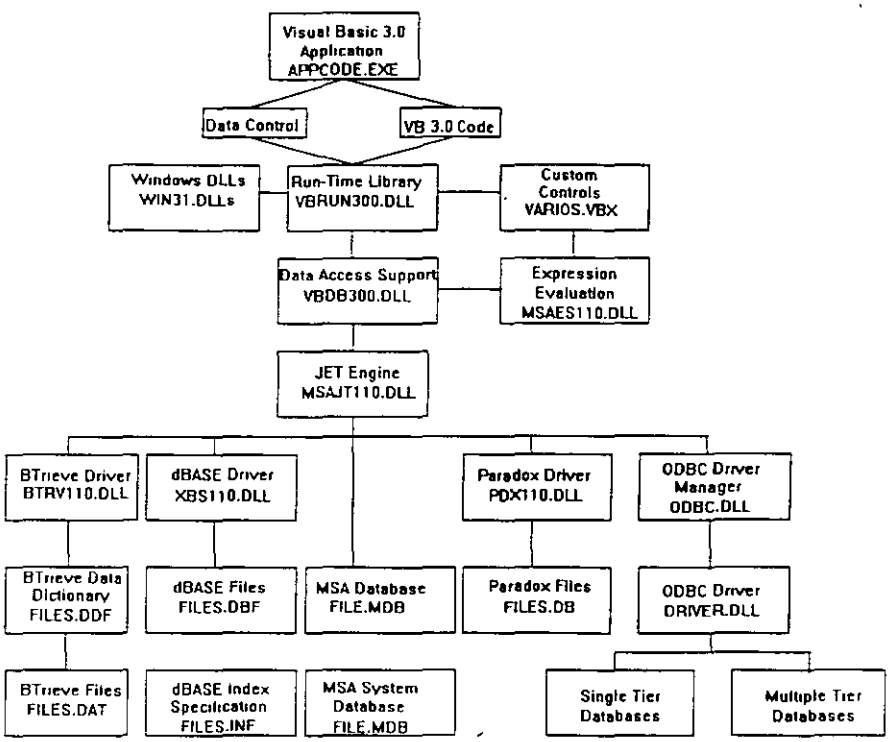


Fig. 4.1

4.2 Uso de Bases de Datos de Access con Visual Basic

Visual Basic está fuertemente orientado hacia las bases de datos .MDB de Access: éste es el tipo de bases de datos "nativo" de Visual Basic .Usted no necesita un controlador extra para crear objetos Database de Access. A diferencia de Access 1.x, que solo soporta archivos .MDB como colecciones de tablas o bases nativas, Visual Basic ocupa las bases de datos por igual. A excepción de los conjuntos de características específicas ofrecidos por los controladores ISAM individuales para otras bases de datos, los objetos Database creados por Visual Basic tienen un conjunto común de colecciones y comparten muchas propiedades y métodos. Access 1.x requiere que usted conecte tablas de bases de datos externas, incluyendo tablas en bases de datos conectadas con ODBC API, a una base de datos de Access y que usted tenga activa solo una base de datos .MDB a la vez. Visual Basic no impone esas limitantes.

La estructura de los archivos de Bases de Datos de Access

Además del archivo DATABASE.MDB, las bases de datos de Access tienen los siguientes archivos opcionales:

Un archivo de llaves (DATABASE.LDB). Cuando usted abre un archivo de bases de datos de Access por primera vez. la base de datos de Access crea un archivo de llaves con el mismo nombre que el archivo de bases de datos, pero con la extensión .LDB. éste archivo contiene referencias binarias a las páginas cerradas por su aplicación y otros usuarios del archivo.

Un archivo SYSTEM.MDA. Las bases de datos aseguradas de Access requieren este archivo. éste archivo contiene los nombres de los usuarios y grupos de usuarios en la tabla del sistema MSysAccounts.

Los desarrolladores profesionales de Access usualmente crean dos tablas para cada aplicación. Una base de datos (la base de datos tabla) contiene los datos en tablas de Access y la otra base de datos (la base de datos aplicación) contiene otros objetos de base de datos que usted puede crear con Access consultas, formas, reportes, macros y módulos (objetos aplicación). Las tablas en la base de datos Tabla están conectadas a la base de datos Aplicación. Este método permite a los desarrolladores cualquiera o todos los objetos de la aplicación simplemente copiando una nueva base de datos aplicación sobre la ya existente. La alternativa es un laborioso proceso de borrado y después importación de los objetos de la aplicación actualizados en una base de datos sencilla.

Cuando usted crea un nuevo archivo .MDB. el mecanismo de bases de datos de Access generalmente reserva 64K de espacio en disco. La mayoría de este espacio esta vacio cuando se genera la base de datos. El contenido inicial de las bases de datos es una colección de tablas del sistema. Una vez que usted llena el espacio vacío restante con datos en las tablas. Access automáticamente expande el espacio en bloques de 2K. el tamaño de un cluster de un disco duro formateado convencionalmente para acomodar 2K nuevos de datos en páginas de índices. A diferencia de muchas bases de datos cliente-servidor. usted no necesita reservar espacios en disco adicionales para aumentar los requerimientos de espacio en disco

4.3 Uso de Otros Tipos de Bases de Datos Soportadas por Visual Basic

Los tres controladores ISAM de bases de datos provistos con Visual Basic (XBS110.DLL, PDX110.DLL y BTRV110.DLL). le permiten conectarse con archivos de dBASE III y IV, FoxPro 2.x. Paradox 3.x y Btrieve 5.1+. El mecanismo de bases de datos de Access y los tres controladores de ISAM le permiten emplear archivos que son usados por lo menos por el 90% de las aplicaciones de bases de datos de escritorio actuales. Cada uno de esos tipos de bases de datos foraneos tiene conexión con métodos que difieren.

Bases de Datos de dBASE y FoxPro

Para utilizar bases de datos de dBASE y FoxPro con el mecanismo de bases de datos de Access. sus archivos VB.INI o APPNAME.INI deben contener al menos alguna de las

entradas que identifiquen la ruta y el nombre del controlador ISAM para el tipo de bases de datos:

```
[Installable ISAMs]
dBASE III=d:\path\xbs110.dll
dBASEIV=d:\path\xbs110.dll
FoxPro 2.0= d:\path\xbs110.dll
FoxPro 2.5= d:\path\xbs110.dll
```

Los valores del nombre de elemento que preceden al signo igual en el ejemplo anterior, corresponden exactamente al valor strDBType (sin el punto y coma) que debe ser incluido en la cedena de conexión de su sentencia OpenDatabase().

Para abrir una base de datos de dBASE III en modo compartido para operaciones de lectura-escritura con los archivos .DBF y .NDX que constituyen la base de datos en su directorio D:\DBASE, use las siguientes sentencias:

```
Dim dbXBase As Database
Dim strDBPath As String
Dim strDBType As String
strDBPath = "d:\dbase"
strDBType = "dBASE III"
Set dbXbase = OpenDatabase(strDBPath, False, False, strDBType)
```

Una vez que usted ha abierto la base de datos dbXBase puede obtener y establecer los valores de las propiedades y aplicar cualquiera de los métodos de acceso a datos que son aplicables a las bases de datos de Access.

Bases de Datos de Paradox

Para utilizar bases de datos de dBASE y FoxPro con el mecanismo de bases de datos de Access, sus archivos VB.INI o APPNAME.INI deben contener al menos alguna de las entradas que identifiquen la ruta y el nombre del controlador ISAM para el tipo de bases de datos de Paradox:

```
[Installable ISAMs]
Paradox 3.x=d:\path\pdx110.dll
```

Para abrir una base de datos de Paradox 3.x en modo compartido para operaciones de lectura-escritura con los archivos en su directorio D:\PARADOX, use las siguientes sentencias:

```
Dim dbPdox As Database
Dim strDBPath As String
Dim strDBType As String
```

```
strDBPath = "d:\paradox"
strDBType = "Paradox"
Set dbXbase = OpenDatabase(strDBPath, False, False, strDBType)
```

Paradox aporta la especificación del campo llave primaria para todas las tablas de Paradox que usted genera. Paradox automáticamente crea un índice único en el campo llave primaria de la tabla El nombre del archivo índice de llave primaria para FILENAME.DB es FILENAME.PX. SI su tabla Paradox fue creada con un índice de llave primaria, el archivo índice debe estar localizado en el directorio de su base de datos. Si éste archivo índice no esta o está localizado en otro directorio, no será posible abrir la tabla.

Bases de Datos Btrieve

El abrir una base de datos Novell Btrieve es un proceso mas complejo que abrir una base de datos de xBASE o Paradox, por lo que no nos enfocaremos a bases de datos de este tipo en este manual: sin embargo, a continuación se mencionan los requerimientos para utilizar las bases de datos de Btrieve con Visual Basic:

- Debe tener una copia de la librería WBTRCALL.DLL de Btrieve para Windows localizada en su directorio \WINDOWS o \WINDOWS\SYSTEM. WBTRCALL.DLL no es provista con Visual Basic.

- Para cada base de datos de Btrieve que abra, necesita dos diccionarios de datos de Btrieve, FILE.DDF y FIELD.DDF, en el mismo directorio que el archivo de bases de datos FILENAME.DAT de Btrieve especificado en FILE.DDF.

- Su archivo de Btrieve debe ser de la versión 1.5 o posterior.

- Si esta utilizando Btrieve en un ambiente multiusuario, necesitará el archivo de transacciones BTRIEVE.TRN para los demás usuarios del archivo.

- No es posible mantener índices de Btrieve que tengan los establecidos los atributos Manual o Null. El mecanismo de bases de datos de Access no soporta valores de tipo Null en índices.

- Al igual que otras bases de datos foráneas, necesitará añadir la ruta y el nombre del controlador ISAM para la bases de datos en la sección [Installable ISAMs] de sus archivos de inicialización VB.INI o APPNAME.INI

4.4 Creación de Recordset y Snapshots Mediante Consultas

Visual Basic solo puede crear objetos QueryDef en bases de datos de Access. Los objetos QueryDef son persistentes por lo que necesitan se almacenados en un archivo. No hay otra forma de almacenar objetos bases de datos mas que utilizar bases de datos de Access. Aún si usted esta usando una base de datos de Access no querrá utilizar espacio en disco cada vez que ejecute un QueryDef. Usted puede crear objetos impersistentes alimentando la sentencia SQL que define la consulta como el argumento opcional strSQL de los métodos CreateDynaset() o CreateSnapshot(). Las consultas impersistentes no son solo una opción cuando usted abre tablas de dBASE. FoxPro. Paradox. Btrieve o ODBC como objetos Database.

Objetos Snapshot

Un objeto Snapshot creado desde un objeto Tabla es equivalente al tradicional SQL VIEW de una tabla entera. Cuando usted crea un objeto snapshot de una tabla, una imagen de los datos en la tabla es transferido al buffer de cache de la memoria de su computadora. El proceso de transferir datos del objeto persistente tabla de la base de datos a la instancia del objeto impersistente Snapshot es llamado populrmente Snapshot. Cuando usted posiciona el apuntador al registro al final del archivo (EOF =True), su objeto Snapshot contiene todos los registros de la tabla de la que surgió.

Si su computadora no tiene suficiente RAM para contener al objeto Snapshot. o el tamaño del objeto Snapchot es m.ayor que el que esta establecido por la entrada MaxBufferSize en la sección [ISAM] del archivo VB.INI o APPNAME.INI. los datos que no son almacenados en RAM. serán almacenados en un archivo temporal del disco. La localización del archivo temporal es determinada por la variable de ambiente SET TEMP = d:\path en su archivo AUTOEXEC.BAT. Asi. todos los datos de su objeto Snapshot son almacenados localmente. ya sea en RAM o en Ram y disco. Un Snapshot, por definición. es un objeto de sólo-lectura y no refleja los cambios hechos a la base de datos de la que surge una vez que fué creado.

Si usted no necesita actualizar valores de un conjunto de resultados de una consulta. el crear un snapshot es usualmente un proceso más rápido que crear un Dynaset. La razón de la mejora en el desempeño es que el mecanismo de bases de datos de Access no necesita tener en cuanta el cerrado de las páginas de llaves. Usted también puede mejorar el desempeño de su aplicación limitando el número de columnas a desplegarse en la consulta al estrictamente necesario. De la misma forma. incrementará el desempeño si abre sus objetos Database en modo de sólo-lectura en lugar de crear Snapshot para todas sus consultas.

Objetos RecordSet

Los **DataControl** al igual que el resto de los objetos **RecordSet** muestran solo un registro a la vez. permitiendo posicionarse en diferentes registros. Para tal fin se dispone de tres funciones básicas de movimiento que equivalen a los cuatro botones del **DataControl**:

```
Data1.Recordset.MoveFirst
Data1.Recordset.MovePrevious
Data1.Recordset.MoveNext
Data1.Recordset.MoveLast
```

Ejemplo de lo anterior es son las siguientes intrucciones:

```
Sub Button1_Click ()
        Data1.Recordset.MoveNext
        If Data1.Recordset.EOF Then
                Data1.Recordset.MoveLast
        End if
End Sub

Sub Button1_Click ()
        Data1.Recordset.MovePrevious
        If Data1.Recordset.BOF Then
                Data1.Recordset.MoveFirst
        End if
End Sub
```

El objetivo de utilizar este tipo de funciones es para que el usuario no vea el DataControl y para implementar una serie de mecanismos de seguridad que permitan al usuario avanzar o retroceder en función de variables de programas.

De este modo si especificamos dentro del Evento Load de la Form la instrucción Data1.Visible = False. se ocultara el DataControl. de manera que solo el programa podrá manejarlo y el usuario deberá interacturar con el DataControl exclusivamente a través de los botones y controles que se hayan dispuestos a tal efecto.

Obtener el valor de un campo

Esto se hace a través de la colección Fields. a través de Recordset(campo) o a través del símbolo "!" el cual actúa como separador sintáctico entre registro y campo. Son expresiones validas las siguientes:

```
Data1.Recordset("Apellido")
```

```
Data1.Recordset!"Apellido"
Data1.Recordset.Fields(2).value
```

Pero puede surgir la posibilidad de asignar a una variable alfanumérica el valor Null. entonces para tratar tal posibilidad y evitar un error en tal caso se debe realizar lo siguiente:

```
If Isnuli(Data1.Recordset.fields(2).value) Then
        A$ = " "
Else
        A$ = Data1.Recordset.fields(2).value
End if
```

4.5 Navegación de un Recordset

Recordemos que un **recordset** es un conjunto de registros que asignamos a una variable para poder tratarlos y hacer operaciones con los registros que la componen.

Para asignar un conjunto de registros a un **recordset** utilizábamos la siguiente sentencia:

```
set autores = mibase.openrecordset("authors", dbOpenTable)
```

En este caso hemos utilizado la constante **dbOpenTable**, con lo que le estamos diciendo que asigne al **recordset** autores todos los campos de la tabla authors, que deberá ser una tabla incluida en la base de datos mibase.

Las demás constantes que podemos introducir son **dbOpenDynaset** y **dbOpenSnapShot** y las características de todas ellas son las siguientes:

dbOpenTable: Sólo permite que se asigne a un recordset el contenido completo de una tabla con todos sus campos y registros. Podemos realizar operaciones de lectura, modificación y escritura sobre los registros. Es el método más rápido para realizar modificaciones en una única tabla.

dbOpenDynaset: Permite que se incluyan varios campos de varias tablas, también permite que se asignen los registros que cumplan ciertas condiciones y no necesariamente todos los que componen las tablas como ocurría con el método anterior. Para indicar los campos y registros que queramos se utiliza la sentencia **SELECT** del lenguaje **SQL** encerrada entre comillas en lugar de introducir únicamente el nombre de una tabla:

```
"SELECT nombre, apellidos, departamento FROM profesores, departamentos WHERE
codprof=coddep AND coddep=3 ORDER BY nombre"
```

Esta sentencia **SELECT** obtiene el nombre, los apellidos y el departamento de todos los profesores que pertenezcan al departamento 3 ordenados alfabéticamente por el nombre.

El método **dbOpenDynaset** permite que se realicen operaciones de lectura, modificación y escritura aunque es más lento que el modo **dbOpenTable.**

dbOpenSnapShot: Este modo de abrir recordsets es parecido a **dbOpenDynaset**, con la diferencia de que la única operación permitida sobre sus registros es la de lectura, no permitiendo operaciones de modificación ni escritura. Por tanto este modo es más rápido que el anterior y *lo utilizaremos cuando sólo nos interese realizar una vista o consulta de un conjunto de tablas*. Para asignar el conjunto de registros al **recordset** utilizaremos también una sentencia **SELECT.**

Un objeto Recordset representa los registros de una tabla base o los registros que se generan al ejecutar una consulta.

Comentarios

Utilice los objetos Recordset para manipular datos en una base de datos al nivel de registro. Cuando utiliza objetos de acceso de datos, interactúa con los datos prácticamente utilizando objetos **Recordset**. Todos los objetos **Recordset** se construyen utilizando registros (filas) y campos (columnas). Existen tres tipos de objetos **Recordset**:

- **Recordset de tipo Table**: una representación en código de una tabla base que puede utilizarse para añadir, cambiar o eliminar registros desde una única tabla de base de datos (sólo espacios de trabajo Microsoft Jet).
- **Recordset de tipo Dynaset:** el resultado de una consulta cuyos registros pueden actualizarse. Un objeto **Recordset** de tipo **Dynaset** es un conjunto dinámico de registros que puede utilizarse para añadir, cambiar o eliminar registros desde una tabla o tablas subyacentes de una base de datos. Un objeto **Recordset** de tipo **Dynaset** puede contener campos de una o más tablas de una base de datos. Este tipo corresponde a un cursor de tipo **keyset** ODBC.
- **Recordset de tipo Snapshot:** una copia estática de un conjunto de registros que puede utilizar para encontrar datos o generar informes. Un objeto **Recordset** de tipo **Snapshot** puede contener campos de una o más tablas de una base de datos pero no se puede actualizar. Este tipo corresponde a un cursor de tipo **static** ODBC.
- **Recordset de tipo Forward-only:** idéntico a un tipo **Snapshot** excepto que no se proporciona ningún cursor. Sólo puede avanzar en los registros. Esto mejora el rendimiento en situaciones donde sólo necesita hacer una pasada sencilla en el conjunto de resultado. Este tipo corresponde a un cursor de tipo **forward-only** ODBC.
- **Recordset de tipo Dynamic:** un conjunto de resultado de una consulta de una o más tablas base en las que puede agregar, cambiar o eliminar registros de una consulta que devuelve filas. Además, también aparecen en el objeto **Recordset** los registros que agregan, eliminan o modifican otros usuarios en la tablas base. Este tipo corresponde a un cursor de tipo **dynamic** ODBC (sólo espacios de trabajo ODBCDirect).

Puede elegir el tipo de objeto **Recordset** que quiere crear usando el argumento tipo del método **OpenRecordset**.

En un espacio de trabajo Microsoft Jet, si no especifica un tipo, DAO intenta crear el tipo de objeto **Recordset** con la mayor funcionalidad disponible, comenzando con tabla. Si no está disponible este tipo, DAO intenta un **Dynaset**, después un **Snapshot** y por último un objeto **Recordset** de tipo **Forward-only**.

En un espacio de trabajo ODBCDirect, si no especifica un tipo, DAO intenta crear el tipo de objeto **Recordset** con la respuesta de consulta más rápida, comenzando con **Forward-only**. Si no está disponible este tipo, DAO intenta un **Snapshot**, después un Dynaset y por último un objeto **Recordset** de tipo **Dynamic**.

Cuando se crea un objeto **Recordset** utilizando un objeto **TableDef** no adjunto, se crean objetos **Recordset** de tipo **Table**. Sólo pueden crearse **Recordset** de tipo **Dynaset** o Snapshot con tablas adjuntas o tablas de bases de datos externas ODBC.

Cuando abre el objeto se agrega automáticamente un nuevo objeto Recordset a la colección Recordsets y se elimina automáticamente cuando lo cierra.

Nota Si utiliza variables para representar un objeto **Recordset** y el objeto **Database** que contiene el conjunto de registros, compruebe que las variables tengan el mismo alcance o duración. Por ejemplo, si establece una variable global que representa un objeto **Recordset**, debe asegurarse de que la variable que represente la base de datos que contiene el conjunto de registros también sea global o se encuentra en un procedimiento **Sub** o **Function** con la palabra clave **Static**.

IMPORTANTE!!! Su aplicación puede crear tantas variables objeto **Recordset** como se necesiten. Un objeto **Recordset** puede hacer referencia a una o más tablas o consultas y los campos sin conflictos.

Los **Recordset** de tipo **Dynaset** y **Snapshot** se almacenan en la memoria local. Si no hay suficiente espacio en la memoria local para almacenar los datos, el motor de base de datos Microsoft Jet guarda los datos adicionales en el disco TEMP. Si este espacio está agotado, se producirá un error.

IMPORTANTE!!! La colección predeterminada de un objeto **Recordset** es la colección **Fields** y la propiedad predeterminada de un objeto **Field** es la propiedad **Value**. El código puede simplificarse utilizando estos valores predeterminados.

Cuando se crea un objeto **Recordset**, el registro activo se coloca como primer registro si existen varios registros. Si no hay registros, el valor de la propiedad RecordCount será 0 y los valores de la propiedad BOF y EOF serán True.

IMPORTANTE!!! Puede utilizar los métodos MoveNext, MovePrevious, MoveFirst y MoveLast para volver a establecer el registro activo. Los objetos Recordset de tipo Forward-only sólo admiten el método MoveNext. Cuando se utilizan los métodos Move para moverse entre los registros (o "andar" a través del objeto Recordset), puede utilizar las propiedades BOF y EOF para comprobar el inicio o el fin del objeto Recordset.

Con los objetos Recordset de tipo Dynaset y Snapshot en un espacio de trabajo Microsoft Jet, también puede utilizar los métodos Find, como FindFirst, para localizar un registro específico basado en un criterio. Si no se encuentra el registro, la propiedad NoMatch se establece a True Para objetos Recordset de tipo Table, puede buscar registros utilizando el método Seek.

La propiedad Type indica el tipo de objeto Recordset creado y la propiedad Updatable indica si puede cambiar los registros del objeto.

La información acerca de la estructura de la tabla base, como los nombres y los tipos de datos de cada objeto Field y cualquier objeto Index, se almacena en un objeto TableDef.

Para hacer referencia a un objeto Recordset en una colección por su número de orden o por el valor de la propiedad Name, utilice cualquiera de los formatos de sintaxis siguientes:

```
Recordsets(0)
Recordsets("nombre")
Recordsets![nombre]
```

Nota: Puede abrir un objeto Recordset del mismo origen de datos o base de datos más de una vez creando nombres duplicados en la colección Recordsets. Debe asignar objetos Recordset a variables de objeto y hacer referencia a ellos por el nombre de variable.

EJEMPLO DE USO DEL RECORDSET

Este ejemplo demuestra los objetos Recordset y la colección Recordsets abriendo cuatro tipos diferentes de Recordsets, enumerando la colección Recordsets de la Database actual y enumerando la colección Properties de cada Recordset.

```
Sub RecordsetX()

    Dim dbsNeptuno As Database
    Dim rstTable As Recordset
    Dim rstDynaset As Recordset
    Dim rstSnapshot As Recordset
    Dim rstForwardOnly As Recordset
    Dim rstBucle As Recordset
    Dim prpBucle As Property

    Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
```

```
With dbsNeptuno

        ' Abre cada tipo de objeto Recordset.
        Set rstTable = .OpenRecordset("Categorías", _
            dbOpenTable)
        Set rstDynaset = .OpenRecordset("Empleados", _
dbOpenDynaset)
        Set rstSnapshot = .OpenRecordset("Compañías de envíos", _
            dbOpenSnapshot)
        Set rstForwardOnly = .OpenRecordset _
            ("Empleados", dbOpenForwardOnly)

        Debug.Print "Recordsets en la colección " & _
            " Recordsets de dbsNeptuno"

        ' Enumera la colección Recordsets.
        For Each rstBucle In .Recordsets

            With rstBucle
```

5. DESARROLLO AVANZADO DE BASE DE DATOS

5.1 Manejo de Integridad Referencial

Proporcionar integridad referencial

La integridad referencial significa que la clave externa de cualquier tabla de referencia siempre debe aludir a una fila válida de la tabla a la que se haga referencia. La integridad referencial garantiza que la relación entre dos tablas continúe sincronizada durante las actualizaciones y eliminaciones.

Por ejemplo, supongamos que la aplicación tiene una tabla de Títulos y una tabla de Editores como se muestra en la siguiente tabla.

Tabla de Títulos	**Tabla de Editores**
Ti_isbn (clave)	pu_id (clave)
Ti_título	pu_nombre
Ti_fechapublicación	pu_dirección
Pu_id (clave externa)	pu_teléfono

La integridad referencial requiere que estas dos tablas estén sincronizadas. Es decir, la identificación de cada editor (pu_id) de la tabla de Títulos también debe aparecer en la tabla de Editores.

La aplicación no puede borrar la fila pu_id de la tabla de Editores porque eso dejaría a pu_id de la tabla de Títulos sin una referencia. No obstante, se podría permitir la eliminación de la fila pu_id de la tabla de Editores y eliminar también cada fila de la tabla de Títulos que tenga el mismo pu_id. Tal hecho mantendría la integridad referencial de estas dos tablas.

De manera similar, la aplicación no puede agregar una fila a la tabla de Títulos sin que exista ya un pu_id válido en la tabla de Editores. Para hacer esto, serían necesarios datos "defectuosos" en el campo pu_id. De manera que, la aplicación debe garantizar una fila válida de Editores antes de insertar la fila Títulos relacionada.

La implementación real de la integridad referencial depende enteramente del motor de almacenamiento de datos que se elija y de los requisitos de diseño de la aplicación. Históricamente, las aplicaciones que utilizan archivos VSAM de grandes sistemas usaban código de la aplicación para manejar la integridad referencial. Actualmente, incluso aunque

la aplicación utilice SQL Server. eso no significa que deba utilizar activaciones. claves externas. restricciones y eliminaciones en cascada para mantener la integridad referencial. Puede optar de nuevo por manejar problemas referenciales con código basado en la aplicación.

5.2 Violaciones a los Datos

Para garantizar la integridad de los datos y la no violación de los mismos debemos seguir las instrucciones que se mencionan a continuación

La integridad de los datos son los valores reales que se almacenan y utilizan en la estructura de los datos de la aplicación. Dicha aplicación debe ejercer un control deliberado sobre cada proceso que utilice los datos con el fin de garantizar la corrección continuada de la información.

Puede garantizar la integridad de los datos a través de la cuidadosa implementación de varios conceptos clave. incluidos:

- La normalización de datos
- El uso de reglas empresariales
- El suministro de integridad referencial
- La validación de los datos

Las siguientes tabla le informará acerca de algunas maneras importantes de garantizar la integridad y seguridad de los datos de la aplicación.

Tema	**Descripción**
Normalizar los datos	Se explica el proceso de ajuste de las definiciones de los datos para eliminar la repetición de grupos y las dependencias innecesarias.
Definir reglas empresariales para el acceso a datos	Se describe cómo las reglas empresariales controlan la manipulación de los datos de la aplicación y cómo otras aplicaciones las reutilizan normalmente.
Proporcionar integridad referencial	Se describe cómo la integridad referencial protege los datos de daños.
Validar los datos	Se describe la comprobación de rango. la validación de campos y formas más complejas de validación de datos.

5.3 Elementos Multiusuario

Un entorno multiusuario es aquél al que otros usuarios pueden conectarse y realizar cambios en la misma base de datos en la que trabaja. Como resultado, muchos usuarios pueden trabajar con los mismos objetos de base de datos al mismo tiempo. De este modo, un entorno multiusuario introduce la posibilidad de que los diagramas de base de datos estén afectados por los cambios realizados por otros usuarios y viceversa. Los cambios pueden incluir cambios a las copias de los diagramas, a los otros diagramas de los usuarios que comparten objetos de base de datos con los diagramas o a la base de datos subyacente.

Un problema clave al trabajar con bases de datos en un entorno multiusuario es el de los permisos de acceso. Los permisos que tiene para la base de datos determinan el alcance del trabajo que puede hacer con la base de datos. Por ejemplo, para realizar cambios a objetos en una base de datos, debe tener los permisos de escritura apropiados para la base de datos. Para obtener más información acerca de los permisos en la base de datos, vea la documentación de la base de datos.

Como uno de los múltiples usuarios, puede necesitar resolver uno de los problemas siguientes al trabajar con el Diseñador de base de datos:

- Propiedad de los objetos de la base de datos
- Diagramas afectados por cambios de otros usuarios
- Objetos de base de datos eliminados por otro usuario

Puede trabajar con diagramas de base de datos en un entorno multiusuario; es decir, un entorno en el que más de un usuario puede realizar cambios a un diagrama de base de datos y la base de datos al mismo tiempo. Cuando guarda un diagrama, el Diseñador de base de datos verifica que la base de datos no se ha modificado desde la última vez que guardó el diagrama. Si otro usuario ha realizado cambios, se le notificará que la base de datos ha sido modificada. Puede necesitar reconciliar estos cambios, tanto en el diagrama de base de datos como en la propia base de datos.

5.4 El Esquema ODBC

Existen tres tipos de bases de datos que se pueden accesar desde Visual Basic:

- Bases nativas de Microsoft Access. Estas bases de datos son accesadas directamente por Visual Basic.
- Bases de datos indexadas con método de acceso secuencial (ISAM). Por ejemplo bases de datos de Dbase, Paradox y Btrieve. Visual Basic obtiene estas

tablas a través de drivers instalados por el usuario que ligan a Visual Basic a bases de datos específicas.

- Bases de datos en sistemas abiertos (ODBC). Estas incluyen sistemas de administración de bases de datos Cliente-Servidor tales como Microsoft SQL Server y Oracle. Visual Basic obtiene estas bases a través de los drivers ODBC correspondientes.

Cuándo se debe utilizar ODBC

Varios factores influyen en la elección de la aproximación ODBC. Incluyen la necesidad de alto rendimiento, más control granular sobre la interfaz y una pequeña huella.

La API de ODBC es considerablemente más difícil de programar que las interfaces basadas en objetos, pero proporciona un control más sensible del origen de datos. A diferencia de otras tecnologías de acceso a datos (como ADO, RDO u ODBCDirect), la API de ODBC no está hecha "a prueba de balas". Aunque es bastante fácil producir errores de ODBC durante la programación, la API del ODBC proporciona un control de errores excelente con mensajes de error detallados. En general, programar, depurar y dar soporte a una aplicación basada en la API de ODBC requiere muchos conocimientos, experiencia y muchas líneas de código. Como regla general, los programadores prefieren tener acceso a datos mediante la utilización de una interfaz de objetos de más alto nivel y más sencilla, como ADO.

ODBC no es apropiado para datos no relacionales, como ISAM (Indexed Sequential Access Method) porque no tiene interfaces para buscar registros, establecer intervalos o examinar índices. ODBC no se ha diseñado para tener acceso a datos de tipo ISAM. Aunque puede utilizar el controlador ODBC de Microsoft Jet para controlar datos de ISAM y datos nativos del motor Microsoft Jet, lo que realmente pasa es que el motor de base de datos Microsoft Jet convierte los datos de ISAM a datos relacionales y proporciona funcionalidad limitada de tipo ISAM. El rendimiento en esta situación es lento debido a la capa adicional impuesta por el motor Microsoft Jet.

Si su aplicación requiere un acceso muy rápido a datos ODBC existentes y desea escribir muchas líneas de código complejo (o ya tiene un lote de código ODBC disponible para reutilizarlo), ODBC es una buena elección.

6. CREACION DE REPORTES IMPRESOS

6.1 Diseño de Reportes

Hay varias maneras de colocar texto y gráficos en el objeto **Printer**. Para imprimir con el objeto **Printer**, siga las instrucciones siguientes:

- Asigne el miembro específico de la colección **Printers** al objeto **Printer** si quiere imprimir en una impresora distinta de la impresora predeterminada.
- Coloque texto y gráficos en el objeto **Printer**.
- Imprima el contenido del objeto **Printer** con los métodos **NewPage** o **EndDoc**. Propiedades del objeto Printer

Las propiedades del objeto **Printer** coinciden inicialmente con las de la impresora predeterminada definida en el Panel de control de Windows. En tiempo de ejecución, puede establecer cualquiera de las propiedades del objeto **Printer**, entre las que se incluyen: **PaperSize**, **Height**, **Width**, **Orientation**, **ColorMode**, **Duplex**, **TrackDefault**, **Zoom**, **DriverName**, **DeviceName**, **Port**, **Copies**, **PaperBin** y **PrintQuality**. Para obtener más detalles y la sintaxis de estos métodos, vea la *Referencia del lenguaje*.

Si la propiedad **TrackDefault** es **True** y cambia la impresora predeterminada en el Panel de control de Windows, los valores de las propiedades del objeto **Printer** reflejarán las propiedades de la nueva impresora predeterminada.

No puede modificar algunas de las propiedades en medio de la impresión de una página una vez establecida la propiedad. La modificación de estas propiedades sólo afectará a las páginas siguientes. Las instrucciones siguientes muestran cómo puede imprimir cada página con una calidad de impresión diferente:

```
For pageno = 1 To 4

   Printer.PrintQuality = -1 * pageno

   Printer.Print "La calidad de esta página es"; pageno

   Printer.NewPage

Next
```

Los valores de la calidad de impresión pueden variar de – 4 a – 1, o un entero positivo correspondiente a la resolución de la impresora en puntos por pulgada (PPP). Por ejemplo, el código siguiente establecería la resolución de la impresora a 300 PPP:

```
Printer.PrintQuality = 300
```

Nota El efecto de los valores de las propiedades del objeto **Printer** depende del controlador suministrado por el fabricante de la impresora. Puede que algunos valores no tengan efecto o que varios valores de propiedades diferentes tengan todos el mismo efecto. Los valores fuera del intervalo permitido puede que produzcan o no un error. Para obtener más información acerca de los controladores concretos, vea la documentación del fabricante.

Propiedades de escala

El objeto **Printer** tiene estas propiedades de escala:

- **ScaleMode**
- **ScaleLeft** y **ScaleTop**
- **ScaleWidth** y **ScaleHeight**
- **Zoom**

Las propiedades **ScaleLeft** y **ScaleTop** definen las coordenadas x e y, respectivamente, de la esquina superior izquierda de una página imprimible. Si modifica los valores de **ScaleLeft** y **ScaleTop**, puede crear los márgenes izquierdo y superior de la página impresa. Por ejemplo, puede usar **ScaleLeft** y **ScaleTop** para centrar un formulario impreso (PFrm) en la página, con estas instrucciones:

```
Printer.ScaleLeft = -((Printer.Width - PFrm.Width) / 2)

Printer.ScaleTop = -((Printer.Height - PFrm.Height) _

   / 2)
```

Muchas impresoras aceptan la propiedad **Zoom**. Esta propiedad define el porcentaje de escala del resultado. El valor predeterminado de la propiedad **Zoom** es 100, que indica que el resultado se imprimirá al 100 por ciento del tamaño (tamaño real). Puede usar la propiedad **Zoom** para reducir o ampliar la página con respecto al papel en el que se imprime. Por ejemplo, si **Zoom** es 50, la página impresa aparece con la mitad de alto y la mitad de ancho de la página de papel. La sintaxis siguiente establece la propiedad **Zoom** a la mitad del tamaño del objeto **Printer** predeterminado:

```
Printer.Zoom = 50
```

Colocar texto y gráficos

Puede establecer las propiedades **CurrentX** y **CurrentY** del objeto **Printer**, de la misma forma que para los formularios y los cuadros de imagen. Con el objeto **Printer**, estas propiedades determinan dónde se coloca el resultado en la página real. Las instrucciones siguientes establecen las coordenadas de dibujo a la esquina superior izquierda de la página actual:

```
Printer.CurrentX = 0
```

```
Printer.CurrentY = 0
```

También puede usar los métodos **TextHeight** y **TextWidth** para colocar texto en el objeto **Printer**. Para obtener más información acerca del uso de estos métodos de texto, vea "Presentar en una ubicación específica", anteriormente en este capítulo.

Imprimir formularios en el objeto Printer

Puede que le interese que la aplicación imprima uno o varios formularios junto con la información de dichos formularios, especialmente si el diseño del formulario corresponde a un documento impreso como una factura o una ficha de hora. La manera más sencilla de hacer esto es mediante el método **PrintForm**. Para obtener la mayor calidad en una impresora láser, utilice el método **Print** y los métodos gráficos con el objeto **Printer**. Recuerde que el uso del objeto **Printer** requiere más tiempo de desarrollo, puesto que tiene que volver a crear el formulario en el objeto **Printer** antes de imprimirlo.

Volver a crear un formulario en el objeto **Printer** también requiere volver a crear:

El esquema del formulario, incluidos el título y las barras de menús.

Los controles y su contenido, incluidos texto y gráficos.

El resultado de los métodos gráficos aplicado directamente al formulario, incluido el método **Print**.

El número de elementos que debe volver a crear en el objeto **Printer** depende de la aplicación y de la parte del formulario que necesite imprimir.

Volver a crear el texto y los gráficos de un formulario

Cuando cree texto y gráficos en un formulario mediante los métodos **Print**, **Line**, **Circle**, **PaintPicture** o **PSet**, puede que desee que una copia de esta resultado aparezca en el objeto **Printer**. La manera más sencilla de conseguirlo es escribir un procedimiento independiente del dispositivo que vuelva a crear el texto y los gráficos.

Por ejemplo, en el procedimiento siguiente se utiliza el método **PaintPicture** para imprimir la propiedad **Picture** de un formulario o de un control en cualquier objeto de resultado, como una impresora u otro formulario:

```
Sub PrintAnywhere (Src As Object, Dest As Object)

   Dest.PaintPicture Src.Picture, Dest.Width / 2, _

      Dest.Height / 2

   If Dest Is Printer Then

      Printer.EndDoc

   End If
```

```
End Sub
```

Después, puede llamar a este procedimiento y pasarle los objetos de origen y de destino:

```
PrintAnywhere MiForm, Printer

PrintAnywnere MiForm, TuForm
```

Para obtener más información Para obtener más información, vea "Print (Método)", "Line (Método)", "Circle (Método)" "Pset (Método)" o "PaintPicture (Método)" en la *Referencia del lenguaje*.

Imprimir los controles de un formulario

El objeto **Printer** puede recibir el resultado del método **Print** y de los métodos gráficos, como los métodos **Line** o **PSet**. Pero no puede colocar controles directamente en el objeto **Printer**. Si la aplicación tiene que imprimir controles, debe escribir procedimientos que vuelvan a dibujar cada tipo de control utilizado en el objeto **Printer** o usar el método **PrintForm**.

Imprimir el contenido del objeto Printer

En cuanto haya colocado el texto y los gráficos en el objeto **Printer**, utilice el método **EndDoc** para imprimir el contenido. El método **EndDoc** avanza la página y envía todo el resultado pendiente a la cola. La *cola* intercepta el trabajo de impresión en su camino hacia la impresora y lo envía al disco o a la memoria, en donde se mantiene hasta que la impresora está preparada para recibirlo. Por ejemplo:

```
Printer.Print "Ésta es la primera línea de texto _

   de una pareja."

Printer.Print "Ésta es la segunda línea de texto _

   de la pareja."

Printer.EndDoc
```

Nota Visual Basic llama automáticamente a **EndDoc** si la aplicación termina sin haberlo llamado de forma explícita.

Crear documentos de múltiples páginas

Cuando imprima documentos grandes, puede especificar en el código dónde quiere que empiece una nueva página, mediante el método **NewPage**. Por ejemplo:

```
Printer.Print "Esta es la página 1."

Printer.NewPage

Printer.Print "Ésta es la página 2."
```

```
Printer.EndDoc
```

Cancelar un trabajo de impresión

Puede terminar el trabajo de impresión actual mediante el método **KillDoc**. Por ejemplo, puede consultar al usuario, a través de un cuadro de diálogo, para determinar si se debe imprimir o terminar un documento:

```
Sub PrintOrNot()
   Printer.Print "Ésta es la primera línea para " & _
      "ilustrar el método KillDoc."
   Printer.Print "Ésta es la segunda línea para " & _
      "ilustrar el método KillDoc."
   Printer.Print "Ésta es la tercera línea para " & _
      "ilustrar el método KillDoc."
   If vbNo = MsgBox("¿Desea imprimir el documento?", _
         vbYesNo) Then
      Printer.KillDoc
   Else
      Printer.EndDoc
   End If
End Sub
```

Si el Administrador de impresión del sistema operativo controla el trabajo de impresión, el método **KillDoc** elimina todo el trabajo enviado a la impresora. Sin embargo, si el Administrador de impresión no controla el trabajo de impresión, puede que la página uno ya se haya enviado a la impresora y no se verá afectada por **KillDoc**. La cantidad de datos enviada a la impresora varía ligeramente de un controlador de impresora a otro.

Nota No puede usar el método **KillDoc** para terminar un trabajo de impresión iniciado con el método **PrintForm**.

6.2 Impresión de Objetos Tipo Recordset

Imprimir un informe de datos

Existen dos maneras de imprimir un informe de datos. En modo Vista preliminar (mediante el método **Show**), el usuario puede hacer clic en el botón **Imprimir** del informe de datos o bien el propio programa puede usar el método **PrintReport** para habilitar la impresión. Si se produce un error durante la impresión, puede interceptarlo con el evento Error.

Cuando imprima un informe por programa, tiene dos opciones: mostrar en pantalla el cuadro de diálogo **Imprimir** o imprimir directamente sin presentar el cuadro de diálogo.

Para mostrar el cuadro de diálogo Imprimir

Agregue un control **CommandButton** a un formulario.

En el evento Click del botón, escriba el código siguiente:

```
DataReport1.PrintReport True
```

El cuadro de diálogo **Imprimir** permite al usuario seleccionar una impresora. imprimir a un archivo. seleccionar un intervalo de páginas o especificar el número de copias para imprimir.

Nota Es necesario tener varias impresoras instaladas en el equipo para poder elegir una de ellas.

Imprimir sin un cuadro de diálogo

En algunos casos es preferible imprimir el informe sin la intervención del usuario. El método **PrintReport** ofrece también la posibilidad de imprimir todas las páginas o sólo un intervalo de páginas seleccionado.

Para imprimir sin mostrar el cuadro de diálogo

Agregue un control **CommandButton** a un formulario.

En el evento Click del botón. escriba el código siguiente:

```
DataReport1.PrintReport False
```

O bien. utilice el código siguiente para especificar un intervalo de páginas para imprimir.

```
DataReport1.PrintReport False, rptRangeFromTo, 1, 2
```

6.3 Uso del método Print

El Diseñador de informe de datos de Microsoft es un generador de informes de datos versátil con capacidad integrada de creación de informes jerárquicos por capas. Puede usarse con un origen de datos como el Diseñador de entorno de datos para crear informes con datos procedentes de muchas tablas relacionales diferentes. Permite imprimir los informes y exportarlos a archivos con formato HTML o de texto.

Posibles usos

- Crear automáticamente informes que se exportan en formato HTML para su distribución instantánea a través de Internet.
- Crear informes diarios con las sumas de transacciones efectuadas cada día.
- Características del Diseñador de informe de datos
- El Diseñador de informe de datos tiene varias características:

Funcionalidad arrastrar y colocar aplicada a campos. Esta funcionalidad permite arrastrar campos del Diseñador de entorno de datos de Microsoft al Diseñador de informe de datos. Cuando utiliza esta técnica, Visual Basic crea automáticamente un control **TextBox** en el informe de datos y establece las propiedades **DataMember** y **DataField** del campo que ha colocado. También puede arrastrar un objeto **Command** del Diseñador de entorno de datos al Diseñador de informe de datos. En este caso, se creará en el informe de datos un control cuadro de texto por cada uno de los campos del objeto **Command** y se asignarán valores apropiados a las propiedades **DataMember** y **DataField** de cada cuadro de texto.

Controles del Cuadro de herramientas. El Diseñador de informe de datos posee un conjunto de controles propios. Cuando agrega un Diseñador de informe de datos al proyecto, los controles se crean automáticamente en una ficha llamada **DataReport** del Cuadro de herramientas. Muchos de estos controles tienen un funcionamiento idéntico a los controles intrínsecos de Visual Basic e incluyen un control **Label**, **Shape**, **Image**, **TextBox**, y **Line**. El sexto control (el control **Function**) genera automáticamente información de uno de los cuatro tipos siguientes: Sum (suma), Average (promedio), Minimum (mínimo) o Maximum (máximo). Para obtener más información acerca del control **Function**, vea "Agregar un control Function al informe de datos".

Vista preliminar. El método **Show** permite mostrar una vista preliminar del informe. El informe de datos se genera y se muestra en su propia ventana.

Nota Es necesario tener una impresora instalada en el equipo para poder usar el modo de vista preliminar.

Impresión de informes. Es posible llamar al método **PrintReport** desde un programa para imprimir un informe. Cuando el informe de datos está en el modo de vista preliminar, los usuarios también pueden imprimirlo si hacen clic en el icono de impresora de la barra de herramientas.

Nota Es necesario tener una impresora instalada en el equipo para poder imprimir un informe.

Exportación de archivos. El método **ExportReport** permite exportar la información del informe de datos. Los formatos de exportación incluyen HTML y texto.

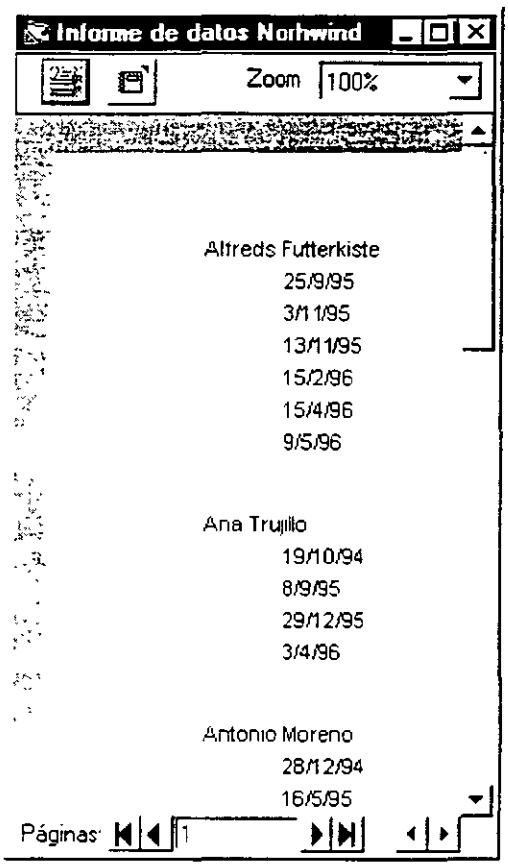
Exportación de plantillas. Es posible crear una colección de archivos de plantilla para usarlos con el método **ExportReport**. Esto es conveniente para exportar informes a varios formatos, cada uno ajustado al tipo de informe.

Operación asíncrona. Los métodos **PrintReport** y **ExportReports** del objeto **DataReport** son operaciones asíncronas. El evento ProcessingTimeout permite supervisar el estado de cada una de estas operaciones y cancelarlas si llevan demasiado tiempo.

Crear un informe de datos sencillo

Este tema muestra cómo puede crear un informe de datos sencillo mediante un Diseñador de entorno de datos utilizado como un origen de datos. El Diseñador de entorno de datos emplea la base de datos NorthWind incluida con Visual Basic para crear un sencillo cursor jerárquico. El cursor contiene dos tablas. Clientes y Pedidos, y ambas están vinculadas mediante el campo IdCliente. El informe terminado es similar a la siguiente figura.

Informe de datos sencillo: Fechas de los pedidos de clientes



Antes de empezar el procedimiento paso a paso, compruebe que la base de datos Northwind (Nwind.mdb) está presente en su equipo. Si no está, copie el archivo del CD de Visual Basic al disco duro.

Para insertar un cursor jerárquico sencillo en el Diseñador de entorno de datos

1. Cree un nuevo proyecto **Standard EXE**.

2. En el menú **Proyecto**, haga clic en **Agregar entorno de datos** para agregar un diseñador al proyecto. Si el diseñador no aparece entre las opciones del menú **Proyecto**, haga clic en **Componentes**. Haga clic en la ficha **Diseñadores** y en **Entorno de datos** para agregar el diseñador al menú.

Nota El menú **Proyecto** proporciona la lista de los cuatro primeros tipos de diseñadores ActiveX cargados para un proyecto. Si se cargan más de cuatro diseñadores, los siguientes aparecerán en el submenú **Más diseñadores ActiveX** del menú **Proyecto**.

3. En el cuadro de diálogo **Data Link Properties**, haga clic en **Microsoft Jet 3.51 OLE DB Provider**. Esta selección del corrige el proveedor OLE DB para tener acceso a la base de datos Jet.

4. Haga clic en el botón **Siguiente** para llegar a la ficha **Conexión**.

5. Haga clic en el botón **Examinar** (...) que se encuentra junto al primer cuadro de texto.

6. Utilice el cuadro de diálogo **Seleccionar base de datos de Access** para desplazarse por el archivo nwind.mdb, que está instalado en el directorio Archivos de programa\Microsoft Visual Studio\Vb98.

7. Haga clic en **Aceptar** para cerrar este cuadro de diálogo.

8. Haga clic con el botón secundario de *mouse* (ratón) en el icono **Connection1**, y elija **Cambiar nombre**. Cambie el nombre del icono a Northwind.

9. Haga clic con el botón secundario del *mouse* en el elemento Norhwind y, después, haga clic en **Agregar comando** para mostrar el cuadro de diálogo **Command1**. En el cuadro de diálogo, establezca las propiedades como se indica a continuación:

Propiedad	**Valor**
Nombre de comando	Clientes
Conexión	Northwind
Objeto de base de datos	Table
Nombre de objeto	Clientes

10. Haga clic en **Aceptar** para cerrar el cuadro de diálogo.

11. Haga clic con el botón secundario del *mouse* (ratón) en el comando **Clientes** y haga clic en **Agregar comando secundario** para mostrar el cuadro de diálogo **Command2**. En el cuadro de diálogo, establezca las propiedades como se indica a continuación:

Propiedad	Valor
Nombre de comando	Pedidos
Conexión	Northwind
Objeto de base de datos	Table
Nombre de objeto	Pedidos

12. Haga clic en la ficha **Relación**. La casilla de verificación **Relacionar con un comando primario** debería estar activada. El cuadro **Primario** debería contener **Clientes**: los cuadros **Campos primarios** y **Campos secundarios y parámetros** deberían mostrar **IdCliente**.

    Cuando se diseñan bases de datos relacionales, se suele usar el mismo nombre para los campos que permiten vincularlas. En este caso, los dos campos vinculados se denominan IdCliente. El Diseñador de entorno de datos hace coincidir automáticamente estos dos campos en el cuadro de diálogo.

13. Haga clic en **Agregar**. Haga clic en **Aceptar** para cerrar el cuadro de diálogo.

    Al hacer clic en el botón **Agregar**, agrega la relación al objeto **Command**. Después de cerrar el cuadro de diálogo, el Diseñador de entorno de datos refleja las relaciones de los dos comandos como una jerarquía. Esta jerarquía se usará para crear el informe de datos.

14. Establezca las propiedades del proyecto y del diseñador según los valores indicados a continuación y, después, guarde el proyecto:

Objeto	Propiedad	Valor
Project	**Name**	**prjNwind**
DataEnvironment	**Name**	**deNwind**
Form	**Name**	**frmShow**

Crear el informe de datos

Después de haber creado el Diseñador de entorno de datos, puede crear un informe de datos. Debido a que no todos los campos del entorno de datos son siempre útiles en un

informe. los siguientes temas muestran cómo puede crear un informe limitado que muestra sólo unos pocos campos.

Para crear un nuevo informe de datos

1. En el menú **Proyecto**. haga clic en **Agregar informe de datos**; Visual Basic agregará el informe de datos al proyecto. Si el diseñador no aparece en el menú **Proyecto**. haga clic en **Componentes**. Haga clic en la ficha **Diseñadores** y en **Informe de datos** para agregar el diseñador al menú.

Nota El menú **Proyecto** proporciona la lista de los cuatro primeros tipos de diseñadores ActiveX cargados. Si se cargan más de cuatro diseñadores. los siguientes aparecerán en el submenú **Más diseñadores ActiveX** del menú **Proyecto**.

2. Establezca las propiedades del objeto **DataReport** según la tabla siguiente:

Propiedad	**Valor**
Name	**rptNwind**
Caption	Informe de datos Northwind

3. En la ventana **Propiedades**. haga clic en **DataSource** y. después. en **deNwind**. A continuación. haga clic en **DataMember** y en **Clientes**.

   Importante Es necesario abrir el Diseñador de entorno de datos antes de establecer la propiedad **DataSource** con el valor **deNwind**. Si el diseñador está cerrado. presione Ctrl-R para mostrar la ventana Proyecto y haga doble clic en el icono del entorno de datos.

4. Haga clic con el botón secundario del mouse (ratón) en el Diseñador de informe de datos y en **Obtener estructura**.

   Acaba de agregar una nueva sección de grupo al diseñador. Cada sección de grupo posee una correspondencia unívoca con un objeto **Command** en el entorno de datos: en este caso. la nueva sección de grupo corresponde al objeto **Command** denominado Clientes. Observe también que al encabezado de grupo corresponde una sección Pie de grupo.

5. Arrastre el campo NombreCompañía (bajo el comando **Clientes**) del Diseñador de entorno de datos a la sección **Encabezado de grupo (Clientes_Encabezado)**.

   La sección Encabezado de grupo puede contener cualquier campo del comando Clientes. pero para facilitar la demostración. este ejemplo sólo muestra por el momento el nombre Clientes.

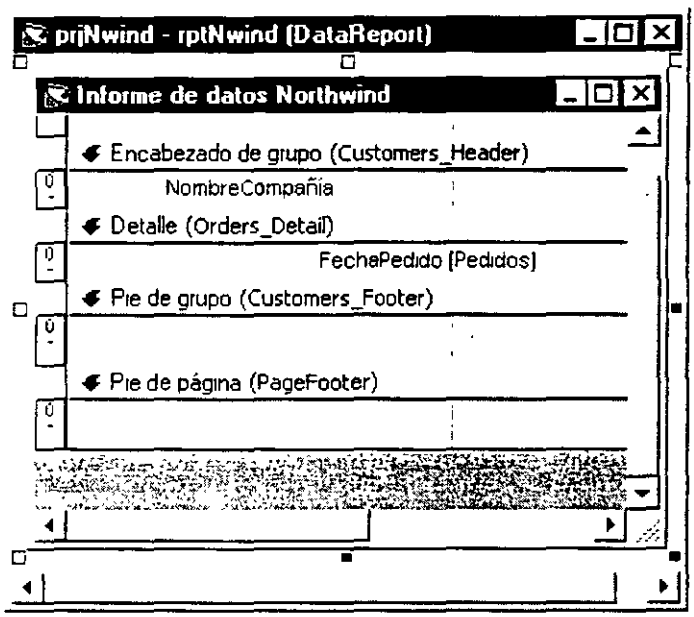
6. Elimine el control **Label** (**rptLabel**) denominado **Label1**.

Si no desea incluir un control **Label** con el control **TextBox**, desactive la opción **Arrastrar y soltar títulos de campo** de la ficha **Asignación de campos** del cuadro de diálogo **Opciones** del Diseñador de entorno de datos.

7. Arrastre el campo **FechaPedido** (bajo el comando **Pedidos**) del Diseñador de entorno de datos a la sección **Detalles de pedidos**. Elimine el control **Label**.

   La sección Detalles de pedidos representa la sección "repetida" más interior y, por tanto, corresponde el objeto **Command** de nivel inferior de la jerarquía del entorno de datos: el objeto **Command** denominado Pedidos.

8. Cambie el tamaño de las secciones del Diseñador de informe de datos para que sea similar a la figura siguiente:



   Es importante que cambie el alto de la sección Detalles de pedidos para que sea lo más corta posible, ya que el alto se multiplicará por cada uno de los objetos FechaPedido devueltos por NombreCompañía. Cualquier espacio innecesario por encima o por debajo del cuadro de texto FechaPedido aumentará la cantidad de espacio sin usar en el informe final.

9. Guarde el proyecto.

Vista preliminar del informe de datos con el método Show

Ahora que se han creado los objetos del entorno de datos y del informe de datos, casi ha llegado el momento de ejecutar el proyecto. Solamente queda un paso: escribir código para mostrar el informe de datos.

Para mostrar el informe de datos en tiempo de ejecución

1. En la ventana **Explorador de proyectos**, haga doble clic en el icono **frmShow** para mostrar el Diseñador de formularios.

2. En **Cuadro de herramientas**, haga clic en la ficha **General**.

   Cuando agrega un Diseñador de informe de datos al proyecto, sus controles se agregan a la ficha denominada **DataReport**. Para usar los controles estándar de Visual Basic, debe cambiar a la ficha **General**.

3. Haga clic en el icono **CommandButton** y dibuje un botón de comando en el formulario.

4. Establezca las propiedades del control **Command1** como se indica en la tabla siguiente:

Propiedad	**Valor**
Name	**cmdShow**
Caption	Mostrar informe

5. En el evento Click del botón, pegue el código siguiente.

```
Private Sub cmdShow_Click()
rptNwind.Show

End Sub
```

6. Guarde y ejecute el proyecto.

7. Haga clic en **Mostrar informe** para mostrar el informe en el modo de vista preliminar.

Configurar el informe de datos como el Objeto inicial:

También puede mostrar el informe de datos sin escribir código.

1. En el menú **Proyecto**, haga clic en **Propiedades de prjNwind**.

2. En el cuadro **Objeto inicial**, seleccione **rptNwind**.

3. Guarde y ejecute el proyecto.

Nota Si utiliza este método, puede quitar el objeto **Form** del proyecto.

7. USO DE LIBRERÍAS DE LIGAS DINÁMICAS (DLL'S)

7.1 El propósito de las DLL's

La mayoría de las aplicaciones complejas de Windows usan una estructura de capas comprimida en un archivo ejecutable (.EXE) y una o más librerías dinámicas ligadas (DLL's) que la aplicación llama cuando las necesita.

Los archivos ejecutables de Visual Basic que usted "compila" contienen el equivalente de código objeto de Visual Basic para las formas y código fuente de su aplicación. La principal diferencia entre librerías DOS y Windows es que el código objeto en librerías para las aplicaciones DOS se convierte en una parte permanente del archivo .EXE (llamadas ligas estáticas). mientras que las DLLs de Windows son ligadas en tiempo de ejecución (llamadas librerías dinámicas). Una de las ventajas de las ligas dinámicas es que una simple .DLL. tal como VBRUN300.DLL. puede servir a todas las aplicaciones de Visual Basic en su computadora. Si usted ejecuta mas de un aplicación de Visual Basic al mismo tiempo. Windows crea instancias adicionales de VBRUN300.DLL en memoria para servir a las otras aplicaciones.

La figura 7.1 ilustra la estructura de capas de una aplicación de base de datos de Visual Basic. La siguiente lista explica la lista de eventos que ocurren cuando su aplicación llama a una operación de acceso de datos .usando la terminología de la figura 7.1:

- El archivo .EXE de su aplicación envía un requerimiento a VBRUN300.DLL para realizar una operación en un objeto de acceso a datos.

- VBRUN300.DLL pasa el requerimiento de acceso a datos al DLL de Soporte de Acceso a Datos. VBDB300.DLL.

- VBDB300.DLL traduce el requerimiento a llamadas a función que son compatibles con las llamadas a funciones de la librería JET Engine. MSAJT110.DLL. ésta determina el tipo de base de datos que corresponde al objeto. abre una instancia del controlador de base de datos apropiado y pasa la llamada al controlador. El controlador devuelve el dato (suponiendo que la petición fue para alimentar valores contenidos en los campos de la tabla) respaldando la cadena de DLLs a su aplicación.

- Si su aplicación utiliza en ODBC API. otra capa (el ODBC.DLL) es interpuesto entre el DLL JET Engine y el controlador ODBC que se conecta a la base de datos.

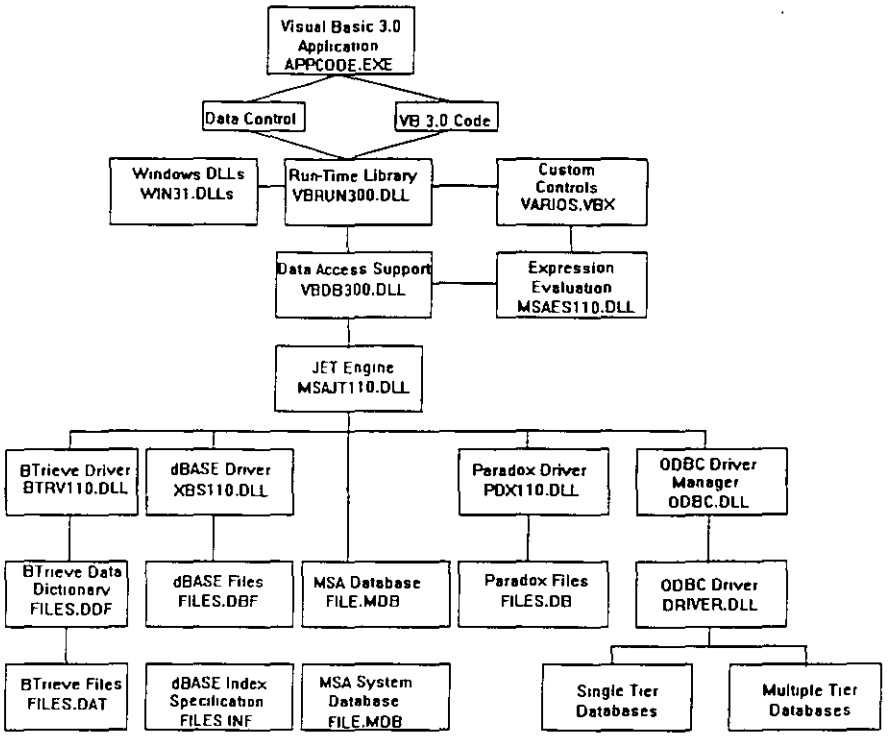


Fig. 7.1

7.2 Implementando llamadas de las Funciones en las DLL's

Usar un procedimiento de DLL en una aplicación

Puesto que los procedimientos de los archivos DLL residen en archivos externos a la aplicación de Visual Basic, debe especificar dónde están ubicados los procedimientos e identificar los argumentos con los que se deben llamar. La instrucción **Declare** proporciona esta información. Una vez declarado un procedimiento de DLL, podrá usarlo en el código como si fuera un procedimiento nativo de Visual Basic.

Importante Al llamar directamente a cualquier DLL desde Visual Basic, perderá las características de seguridad incorporadas del entorno de Visual Basic. Esto significa que aumentará el riesgo de fallos del sistema durante la comprobación o la depuración del código. Para minimizar este riesgo, debe prestar mucha atención a la forma en que declara los procedimientos del archivo DLL, pasa los argumentos y especifica los tipos. En todos los casos, guarde a menudo su trabajo. Las llamadas a los archivos DLL le ofrecen una eficacia excepcional, pero acusan los errores más que otras tareas de programación.

En el ejemplo siguiente se muestra cómo debe llamar a un procedimiento desde la API de Windows. La función llamada **SetWindowText** modifica el título de un formulario. En la

práctica. siempre se modifican los títulos con la propiedad **Caption** de Visual Basic. pero este ejemplo ofrece una manera sencilla de declarar y llamar a un procedimiento.

Declarar un procedimiento de DLL

El primer paso consiste en declarar el procedimiento en la sección Declaraciones de un módulo:

```
Private Declare Function SetWindowText Lib "user32" _
Alias "SetWindowTextA" (ByVal hwnd As Long, _
ByVal lpString As String) As Long
```

Puede encontrar la sintaxis exacta de un procedimiento mediante la aplicación Visor de API o en el archivo Win32api.txt. Si coloca la instrucción **Declare** en un módulo de formulario o de clase. debe ir detrás de la palabra clave **Private**. Sólo puede declarar un procedimiento de DLL por proyecto: después podrá llamarlo todas las veces que desee.

Llamar a un procedimiento de DLL

Después de declarar la función. la llamará de la misma forma en que llamaría a una función estándar de Visual Basic. En este ejemplo se ha adjuntado el procedimiento al evento Form Load:

```
Private Sub Form_Load()
   SetWindowText Form1.hWnd, "Bienvenidos a VB"
End Sub
```

Al ejecutar este código. la función usará primero la propiedad **hWnd** para identificar la ventana cuyo título desea modificar (Form1.hWnd) y luego cambiará el texto de ese título a "Bienvenidos a VB".

Recuerde que Visual Basic no puede comprobar que está pasando los valores correctos a un procedimiento de DLL. Si pasa valores que no son correctos. el procedimiento puede fallar. lo que puede hacer que se interrumpa la aplicación de Visual Basic. Si esto ocurre. debe volver a cargar la aplicación y volver a iniciarla. Tome las debidas precauciones cuando experimente con procedimientos de DLL y guarde su trabajo a menudo.

Nota Pocas llamadas a la API reconocen el tipo de datos **Variant** predeterminado. Sus llamadas serán más robustas si declara variables de tipos específicos y utiliza **Option Explicit**.

8. CREACIÓN DE ARCHIVOS DE AYUDA PARA WINDOWS

8.1 Creación del Texto de Ayuda

No importa la calidad de la interfaz de usuario; habrá veces en que un usuario necesite asistencia. El modelo de asistencia al usuario de su aplicación incluye elementos como la Ayuda en pantalla y la documentación impresa: también puede contener elementos de asistencia al usuario como Información sobre herramientas, barras de estado, ayuda del tipo ¿Qué es esto? y asistentes.

8.2 Creación del Archivo de Proyecto

El modelo de asistencia al usuario debe diseñarse como cualquier otra parte de la aplicación: antes de empezar el desarrollo. El contenido del modelo variará dependiendo de la complejidad de la aplicación y de la audiencia prevista.

Ayuda y documentación

La Ayuda en pantalla es una parte importante de cualquier aplicación; normalmente es el primer lugar en el que el usuario busca cuando tiene alguna duda. Incluso una aplicación sencilla debe proporcionar Ayuda; no proporcionarla es como asumir que los usuarios nunca tendrán dudas o preguntas.

Al diseñar el sistema de Ayuda, tenga en cuenta que su propósito principal es responder a preguntas. Intente pensar como el usuario cuando cree nombres de temas y entradas de índice; por ejemplo, "¿Cómo dar formato a una página?" en lugar de "Menú Edición, Formato de página" hará que el tema sea más fácil de encontrar. No olvide la ayuda contextual; la mayoría de los usuarios encuentra desalentador presionar la tecla F1 para obtener Ayuda acerca de un campo concreto y ver que se les presenta el tema Contenido.

La documentación conceptual, ya sea impresa o en disco compacto, es útil para todas las aplicaciones menos las más sencillas. Puede proporcionar información que sea difícil de transmitir en un tema de Ayuda más corto. Como mínimo, debe proporcionar documentación en forma de archivo Léame que el usuario pueda imprimir si lo desea.

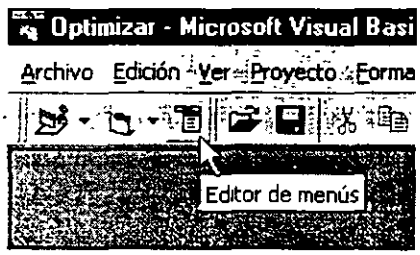
Elementos de asistencia al usuario

Dentro de la interfaz de usuario, hay varias técnicas para proporcionar asistencia al usuario. Visual Basic facilita la inclusión en sus aplicaciones de Información sobre

herramientas. ayuda del tipo ¿Qué es esto?, presentación de estados y asistentes. Usted decide cuáles son los elementos apropiados para su aplicación.

Información sobre herramientas

La información sobre herramientas (ver figura) es una excelente manera de presentar información a los usuarios mientras recorren la interfaz de usuario. Información sobre herramientas es una pequeña etiqueta que se presenta cuando el puntero del *mouse* se mantiene durante un cierto periodo de tiempo sobre un control y que normalmente contiene una descripción de la función del control. Por lo general, se utilizan de forma conjunta con las barras de herramientas; Información sobre herramientas también funciona bien casi en cualquier parte de la interfaz.



La mayoría de los controles de Visual Basic contienen una propiedad para presentar Información sobre herramientas: **ToolTipText**. El siguiente código implementaría una Información sobre herramientas para un botón de comando llamado cmdPrint:

```
cmdPrint.ToolTipText = "Imprime el documento actual"
```

Como con otras partes de la interfaz, asegúrese de que el texto transmita claramente el mensaje deseado al usuario.

Ayuda de tipo ¿Qué es esto?
La ayuda ¿Qué es esto? proporciona un vínculo con un tema de Ayuda emergente (ver la figura) cuando el usuario selecciona ayuda ¿Qué es esto? y hace clic en el cursor ¿Qué es esto? de un control. La ayuda ¿Qué es esto? puede iniciarse desde un botón de una barra de herramientas, un elemento de menú o un botón de la barra de título de un cuadro de diálogo.

Botón Ortografía (barra de herramientas Estándar)

Comprueba la ortografía del documento activo, incluyendo el texto de encabezados, pies de página, notas al pie, notas al final y anotaciones

8.3 Creación del Archivo de Ayuda

Microsoft Visual Basic 6.0 incluye dos aplicaciones Programa de Ayuda que permiten crear temas de Ayuda en formato Ayuda de Windows (WinHelp) o en formato HTML.

WinHelp

Es posible usar el Programa compilador de Ayuda incluido con Microsoft Visual Basic 6.0 para crear archivos de Ayuda en pantalla que presentan texto bidireccional. He aquí sugerencias adicionales para la creación de archivos de Ayuda.

Los archivos creados sólo mostrarán correctamente el texto bidireccional cuando se ejecuten en un entorno Microsoft Windows bidireccional de 32 bits.

Para generar un archivo de Ayuda en pantalla que muestra texto bidireccional: debe crear primero un archivo con formato .RTF con texto y atributos de formato bidireccionales. por ejemplo párrafos escritos de derecha a izquierda. Ciertos procesadores de texto con características bidireccionales agregarán automáticamente esta información a los archivos con formato .RTF que crean. Microsoft Word para Windows 95. Edición Árabe o posterior. o cualquier otra versión para Oriente Medio de Microsoft Word para Windows 95 son ejemplos de productos con esta característica.

Si crea un archivo de Ayuda en pantalla que vaya a ejecutarse en una plataforma que no sea una versión bidireccional de Microsoft Windows de 32 bits. asegúrese de especificar la opción de idioma LCID apropiada en el archivo de proyecto de Ayuda. Por ejemplo. si el archivo va a ejecutarse en la versión de Windows 95 para Estados Unidos. elija Inglés como el idioma para el archivo de Ayuda.

Para cada archivo .RTF compilado con el Programa compilador de Ayuda. todos los textos con el juego de caracteres latino escritos de derecha a izquierda tendrán la misma apariencia independientemente de los atributos que puedan asignarse durante la modificación del archivo Esta limitación hace que los atributos de la fuente Alfabeto latino predeterminado (por ejemplo. la fuente y el tamaño de fuente) se asignen al texto del juego de caracteres latino correspondiente. Esta limitación no existe para los párrafos escritos de izquierda a derecha.

El Programa compilador de Ayuda no admite las tablas con dirección de derecha a izquierda. que son una característica estándar de los procesadores de texto como Microsoft Word Árabe. El Programa compilador de Ayuda compilará las tablas escritas de derecha a izquierda como si se tratara de tablas escritas de izquierda a derecha.

Ayuda HTML

El Programa de Ayuda HTML es una herramienta de Ayuda disponible a nivel mundial que permite crear temas de Ayuda en formato HTML, así como convertir archivos WinHelp existentes. El Programa de Ayuda HTML tiene compatibilidad bidireccional integrada; no obstante, para mostrar temas con texto bidireccional, los usuarios deberán usar un explorador de Internet con la característica bidireccional habilitada. Las versiones de Microsoft Internet Explorer 3.x y Microsoft Internet Explorer 4.x que admiten la característica bidireccional están disponibles en los mercados de Oriente Medio. Vea otras fuentes de información externas, como PSS, los archivos Léame y los sitios Web de las subsidiarias para obtener sugerencias especiales acerca del desarrollo de temas de Ayuda en formato HTML.

8.4 Integración de la Ayuda al Programa

Para activar la ayuda del tipo ¿Qué es esto? desde un menú o una barra de herramientas

1. Seleccione el control para el que desee ofrecer ayuda.
2. En la ventana Propiedades, seleccione la propiedad WhatsThisHelpID.
3. Escriba el Id. de contexto del tema de Ayuda emergente asociado.
4. Repita los pasos 1 a 3 para los demás controles.
5. Seleccione el formulario.
6. En la ventana Propiedades, establezca la propiedad WhatsThisHelpID del formulario a True.
7. En el evento Click del menú o del botón de la barra de herramientas, escriba lo siguiente.

   ```
   nombre_formulario.WhatsThisHelp
   ```

   Cuando el usuario haga clic en el botón o en el menú, el puntero del *mouse* adoptará la forma del puntero ¿Qué es esto?

Para activar la Ayuda ¿Qué es esto? en la barra de título de un cuadro de diálogo personalizado, establezca las propiedades WhatsThisButton y WhatsThisHelp del formulario a True.

Presentación de estado

Una presentación del estado también se puede usar para proporcionar asistencia al usuario de manera muy parecida a Información sobre herramientas. La presentación de estado es una buena manera de ofrecer instrucciones o mensajes que no quepan en una Información sobre herramientas. El control de barra de estado incluido en las ediciones Profesional y Empresarial de Visual Basic es útil para presentar mensajes: también se puede usar un control de etiqueta como presentación de estado.

El texto mostrado en una presentación de estado puede actualizarse de dos maneras: en el evento GotFocus de un control o un formulario, o en el evento MouseMove. Si desea usar esta presentación como elemento de aprendizaje, agregue un elemento al menú Ayuda para activar y desactivar su propiedad Visible.

Para presentar información de estado

1. Agregue un control de etiqueta al formulario.

2. Seleccione el control para el que desee presentar un mensaje.

3. Agregue el siguiente código al evento MouseMove (o GotFocus) del control:

   ```
   nombre_etiqueta.Caption = "Escriba el Id. de cliente en este
   campo"
   ```

   Cuando el usuario mueva el *mouse* sobre el control, el mensaje se presentará en el control de etiqueta.

4. Repita los pasos 2 y 3 para los demás controles.

Asistentes

Un asistente es un elemento de asistencia al usuario que guía al usuario paso a paso a través de un procedimiento, trabajando con los datos reales del usuario. Los asistentes se utilizan normalmente para proporcionar asistencia sobre tareas específicas. Ayudan al usuario a realizar una tarea que de otro modo requeriría un largo (y poco deseable) periodo de aprendizaje y también ofrecen información de experto a un usuario que todavía no lo sea.

Las ediciones Profesional y Empresarial de Visual Basic incluyen el Administrador de asistentes, una herramienta para crear asistentes.

8.5 Creación de Discos de Distribución

Archivos de instalación para paquetes estándar

Los archivos de instalación comprenden todos los archivos necesarios para configurar su aplicación estándar en el equipo del usuario. Incluyen los ejecutables de instalación (setup.exe y setup1.exe), la lista de archivos de instalación (Setup.lst) y el programa de desinstalación (st6unst.exe).

Las aplicaciones de Visual Basic diseñadas para ser distribuidas en disquetes, en CD o en una ubicación de red utilizan los mismos archivos de instalación, sin tener en cuenta si utiliza el Asistente de empaquetado y distribución o el Kit de herramientas de instalación con el fin de crear los programas de instalación. Se enumeran estos archivos a continuación.

Nombre de archivo	Descripción
setup.exe	Programa que el usuario ejecuta con el fin de realizar la instalación previa de los archivos necesarios para la instalación de su aplicación en el equipo del usuario. Por ejemplo, el archivo setup.exe instala el archivo setup1.exe, el archivo DLL de tiempo de ejecución de Visual Basic así como otros archivos sin los que no es posible ejecutar el resto del proceso de instalación.
setup1.exe	Programa de instalación de su aplicación de Visual Basic. El Kit de herramientas de instalación genera este archivo ejecutable y el Asistente de empaquetado y distribución lo incluye en el paquete. Puede volver a nombrar este archivo siempre que el nombre nuevo aparezca reflejado en el archivo Setup.lst.
Setup.lst	Archivo de texto que contiene instrucciones de instalación y que enumera todos los archivos que deben instalarse en el equipo del usuario.
Vb6stkit.dll	Biblioteca con varias funciones utilizadas en Setup1.exe.
St6unst.exe	Herramienta de eliminación de la aplicación.

Nota Las aplicaciones diseñadas para ser entregadas por Internet generalmente no utilizan ninguno de estos archivos. Vea "Paquetes de Internet" previamente en este capítulo para obtener más información acerca de los archivos comprendidos en una entrega de Internet.

Dependencias de la aplicación

Con el fin de ejecutar su aplicación, los usuarios finales necesitarán algunos archivos además de los archivos de tiempo de ejecución comunes y los archivos de

instalación especiales. Muchos de estos archivos le parecerán evidentes: el archivo ejecutable, cualquier archivo de datos y cualquier control ActiveX que utilice. Los archivos menos evidentes son los otros archivos dependientes de su proyecto. Por ejemplo, algunos de los controles ActiveX utilizados por su proyecto pueden a su vez necesitar otros archivos. Una de las tareas del Asistente de empaquetado y distribución es determinar la lista completa de los archivos necesarios.