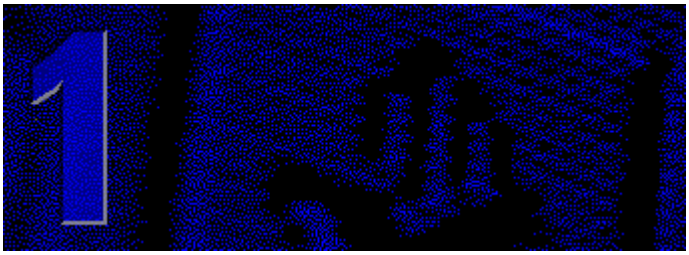
Manual del programador, Parte 1: Programación en Visual FoxPro



Visual FoxPro es una eficaz herramienta de administración de datos, pero además podrá beneficiarse de toda su eficacia para crear aplicaciones. Comprender las técnicas de programación orientada a objetos y el modelo controlado por eventos puede aumentar su productividad como programador.

Capítulo 1 Introducción a la programación

Si está empezando a programar, aprenda el proceso y el método de programación en Visual FoxPro.

Capítulo 2 Programar una aplicación

Cuando programe una aplicación, organice sus componentes con el Administrador de programas, una forma integrada de generar y probar su aplicación a medida que la cree.

Capítulo 3 Programación orientada a objetos

Con la programación orientada a objetos, puede crear componentes de aplicación independientes que respondan a acciones del usuario y al sistema y que se puedan mantener y reutilizar fácilmente.

Capítulo 4 Descripción del modelo de eventos

El modelo de eventos define cuándo y cómo tienen lugar las interacciones con el usuario y el sistema.

Capítulo 1: Introducción a la programación

En Visual FoxPro funcionan juntas la programación por procedimientos y la programación orientada a objetos para permitirle crear aplicaciones potentes y flexibles. Conceptualmente, puede imaginarse que la programación consiste en escribir una secuencia de instrucciones con el fin de realizar tareas específicas. A un nivel estructural, la programación en Visual FoxPro precisa la manipulación de los datos almacenados.

Si no tiene experiencia en programación, este capítulo le ayudará a ponerse en marcha. Si ya conoce otros lenguajes de programación y desea compararlos con Visual FoxPro, vea el tema Visual FoxPro y otros lenguajes de programación. Si desea una descripción de la programación orientada a objetos, consulte el capítulo 3, Programación orientada a objetos.

En este capítulo se abordan los temas siguientes:

Ventajas de la programación

Normalmente, cualquier función que pueda realizar con un programa podrá realizarla también a mano, si dispone de suficiente tiempo. Por ejemplo, si desea consultar información sobre un cliente en una tabla de clientes, como por ejemplo la empresa Ernst Handel, podría hacerlo manualmente si sigue una secuencia concreta de instrucciones.

Para buscar manualmente un único pedido en una tabla

1. En el menú **Archivo**, elija **Abrir**.
2. En el cuadro **Archivos de tipo**, elija **Tabla**.
3. Haga doble clic en Customer.dbf en la lista de archivos.
4. En el menú **Ver**, elija **Examinar**.
5. Desplácese por la tabla, examinando el campo Company de los registros hasta encontrar “Ernst Handel”.

Mediante programación podría conseguir el mismo resultado escribiendo los siguientes comandos de Visual FoxPro en la ventana Comandos:

```
USE Customer
LOCATE FOR Company = "Ernst Handel"
BROWSE
```

Cuando haya localizado el pedido de esta empresa, tal vez desee incrementar la cantidad máxima del pedido en un 3%.

Para incrementar manualmente la cantidad máxima del pedido

1. Presione la tecla Tab para desplazarse hasta el campo `max_ord_amt`.
2. Multiplique el valor mostrado en el campo `max_ord_amt` por 1,03 y escriba el nuevo valor en el campo.

Para conseguir el mismo resultado mediante programación, escriba el siguiente comando de Visual FoxPro en la ventana Comandos:

```
REPLACE max_ord_amt WITH max_ord_amt * 1,03
```

Es relativamente sencillo cambiar la cantidad máxima del pedido para un cliente, ya sea manualmente o escribiendo las instrucciones en la ventana Comandos. Sin embargo, suponga que desea incrementar en un 3% la cantidad máxima de pedido de todos los clientes. Podría hacerlo manualmente, pero le llevaría mucho tiempo y es posible que cometiese errores. Si especifica las instrucciones correctas en un archivo de programa, Visual FoxPro podrá realizar esta tarea con rapidez y facilidad, sin cometer ningún error.

Programa de ejemplo para incrementar las cantidades máximas de pedido de todos los clientes

Código	Comentarios
`USE customer`	Abre la tabla CUSTOMER.
`SCAN`	Examina todos los registros de la tabla y realiza todas las instrucciones comprendidas entre SCAN y ENDSCAN para cada registro.
`REPLACE max_ord_amt WITH ;` `  max_ord_amt * 1.03`	Incrementa la cantidad máxima de pedido en un 3%. (El punto y coma (;) indica que el comando sigue en la línea siguiente).
`ENDSCAN`	Final del código que se ejecuta para cada registro contenido en la tabla.

La ejecución de un programa ofrece numerosas ventajas en comparación con la introducción de distintos comandos en la ventana Comandos:

- Los programas se pueden modificar y volver a ejecutar.
- Se pueden ejecutar programas desde los menús, formularios y barras de herramientas.
- Los programas pueden ejecutar otros programas.

En las siguientes secciones se describe la mecánica, los conceptos y los procesos que subyacen a éste y otros programas de Visual FoxPro.

La mecánica de la programación en Visual FoxPro

Puede programar en Visual FoxPro escribiendo código: instrucciones en forma de comandos, funciones u operaciones que Visual FoxPro puede entender. Puede incluir estas instrucciones en:

- La ventana Comandos.
- Archivos de programa
- Ventanas de código de eventos o de métodos en el Diseñador de formularios o en el Diseñador de clases
- Ventanas de código de procedimientos en el Diseñador de menús
- Ventanas de código de procedimientos en el Diseñador de informes

Usar la ventana Comandos

Puede ejecutar un comando de Visual FoxPro si lo escribe en la ventana Comandos y presiona ENTRAR. Para volver a ejecutar el comando, lleve el cursor a la línea que contiene el comando y presione nuevamente ENTRAR.

Puede ejecutar varias líneas de código en la ventana Comandos como si constituyeran un programa.

Para ejecutar varias líneas de código en la ventana Comandos

1. Seleccione las líneas de código.

2. Presione ENTRAR o elija **Ejecutar selección** en el menú emergente.

Como la ventana Comandos es una ventana de edición, puede modificar comandos con las herramientas disponibles en Visual FoxPro. Puede modificar, insertar, eliminar, cortar, copiar o pegar texto en la ventana Comandos.

La ventaja que supone poder escribir código en la ventana Comandos radica en el hecho de que las instrucciones se ejecutan de inmediato. No es necesario guardar un archivo y ejecutarlo como un programa.

Además, las opciones que elige en los menús y los cuadros de diálogo aparecen en la ventana Comandos como comandos. Puede copiar y pegar estos comandos en un programa de Visual FoxPro y a continuación ejecutar el programa repetidamente, lo cual facilita la ejecución de miles de comandos, una y otra vez.

Crear programas

Un programa de Visual FoxPro es un archivo de texto que contiene una serie de comandos. Puede crear un programa en Visual FoxPro de una de las siguientes maneras:

Para crear un programa

1. En el Administrador de proyectos, seleccione **Programas** en la ficha **Código**.

2. Elija **Nuevo**.

   –O bien–

1. En el menú Archivo, elija Nuevo.

2. En el cuadro de diálogo Nuevo, seleccione Programa.

3. Elija Nuevo archivo.

   –O bien–

- En la ventana **Comandos**, escriba:

  ```
  MODIFY COMMAND
  ```

Visual FoxPro abrirá una nueva ventana denominada Programa1. Podrá entonces escribir su programa en esta ventana.

Guardar programas

Una vez creado un programa, asegúrese de guardarlo.

Para guardar un programa

- En el menú **Archivo**, elija **Guardar**.

Si intenta cerrar un programa sin antes guardarlo, aparecerá un cuadro de diálogo en el que se le preguntará si desea guardar o descartar los cambios realizados en el mismo.

Si guarda un programa creado a partir del Administrador de proyectos, el programa se agregará al proyecto.

Si guarda un programa al que todavía no ha asignado un nombre, se abrirá el cuadro de diálogo Guardar como, en el que podrá especificar el nombre del programa. Cuando haya guardado el programa, podrá ejecutarlo o modificarlo.

Modificar programas

Después de guardar el programa, podrá modificarlo. En primer lugar, abra el programa de una de las siguientes maneras:

Para abrir un programa

- Si el programa forma parte de un proyecto, selecciónelo en el Administrador de proyectos y elija **Modificar**.

  –O bien–

- En el menú **Archivo**, elija **Abrir**. Aparecerá un cuadro de diálogo en el que se muestra una lista de los archivos disponibles. En la lista **Archivos de tipo**, elija **Programa**. En la lista de archivos, seleccione el programa que desea modificar y elija **Abrir**.

  –O bien–

- En la ventana **Comandos**, escriba el nombre del programa que desea modificar:

  ```
  MODIFY COMMAND miprogram
  ```

  –O bien–

- En la ventana **Comandos**, escriba:

  ```
  MODIFY COMMAND ?
  ```

  Cuando aparezca la lista de archivos, seleccione el programa que desea modificar y a continuación elija **Abrir**.

Después de abrir el programa, podrá realizar cambios en el mismo. Cuando haya terminado de introducir los cambios, asegúrese de guardar el programa.

Ejecutar programas

Después de crear un programa, podrá ejecutarlo.

Para ejecutar un programa

- Si el programa forma parte de un proyecto, selecciónelo en el Administrador de proyectos y elija **Ejecutar**.

  –O bien–

- En el menú **Programa**, elija **Ejecutar**. Cuando aparezca la lista de programas, seleccione el programa que desea ejecutar y a continuación elija **Ejecutar**.

  –O bien–

- En la ventana **Comandos**, escriba `DO` y el nombre del programa que desea ejecutar:

  ```
  DO miprogram
  ```

Escribir código en las herramientas de diseño de Visual FoxPro

El Diseñador de formularios, el Diseñador de clases y el Diseñador de menús le permiten integrar fácilmente código de programas mediante la interfaz de usuario, de forma que el código apropiado se ejecute como respuesta a las acciones del usuario. El Diseñador de informes le permite crear informes complejos y personalizados integrando código en el archivo del informe.

Para aprovechar plenamente la eficacia de Visual FoxPro, debe utilizar estas herramientas de diseño. Si desea más información sobre el Diseñador de informes, consulte el capítulo 7, Diseñar informes y etiquetas, del *Manual del usuario*. Para obtener información más detallada sobre el Diseñador de formularios, consulte el capítulo 3, Programación orientada a objetos, de este manual. Para obtener información más detallada sobre el Diseñador de formularios, consulte el capítulo 9, Crear formularios, y si desea más información acerca del Diseñador de menús, consulte el capítulo 11, Diseñar menús y barras de herramientas.

Conceptos básicos de programación

Cuando se programa, se almacenan datos y se manipulan mediante una serie de instrucciones. Los datos y los contenedores en los que se almacenan los datos constituyen la materia prima de la programación. Las herramientas utilizadas para manipular esta materia prima son comandos, funciones y operadores.

Almacenar datos

Los datos con los que trabaja probablemente incluyan períodos de tiempo, dinero y elementos contables, así como fechas, nombres, descripciones, etc. Cada dato corresponde a un determinado tipo, es decir, pertenece a una categoría de datos que se manipula de maneras similares. Podría trabajar directamente con estos datos sin almacenarlos, si bien perdería la mayor parte de la flexibilidad y potencia que ofrece Visual FoxPro. Visual FoxPro aporta numerosos contenedores de almacenamiento con el fin de ampliar su capacidad para manipular fácilmente los datos.

Los tipos de datos determinan la manera en que se almacenan los datos y la forma en que se pueden utilizar tales datos. Puede multiplicar dos números, pero no puede multiplicar caracteres. Puede imprimir caracteres en mayúsculas, pero no puede imprimir números en mayúsculas. En la tabla siguiente se muestran algunos de los principales tipos de datos de Visual FoxPro.

Tipos de datos

Tipo	Ejemplos
Numeric	123 3,1415 – 7
Character	“Prueba” “123” “01/01/98”
Logical	.T. (verdadero) .F. (falso)
Date	{^1998-01-01}
DateTime	{^1998-01-01 12:30:00 p}

Contenedores de datos

Los contenedores de datos le permiten realizar las mismas operaciones con varios datos. Por ejemplo, sumar las horas que ha trabajado un empleado, multiplicarlas por el salario por hora y restar los impuestos para determinar el sueldo que ha percibido el empleado. Deberá realizar estas operaciones para cada empleado y para cada período de pago. Si almacena esta información en contenedores y realiza las operaciones sobre éstos, bastará con sustituir los datos antiguos por los nuevos datos y volver a ejecutar el mismo programa. En la siguiente tabla se enumeran algunos de los principales contenedores de datos disponibles en Visual FoxPro:

Tipo	Descripción
Variables	Elementos individuales de datos almacenados en la memoria RAM (memoria de acceso aleatorio) del PC.
Registros de tabla	Varias filas de campos predeterminados, cada uno de los cuales puede contener un dato definido previamente. Las tablas se guardan en disco.
Matrices	Varios elementos de datos almacenados en la memoria RAM.

Manipular datos

Los contenedores y los tipos de datos le ofrecen los módulos que necesita para manipular los datos. Los elementos finales son los operadores, las funciones y los comandos.

Usar operadores

Los operadores se utilizan para vincular los datos. A continuación se muestran los operadores utilizados habitualmente en Visual FoxPro.

Operador	Tipos de datos válidos	Ejemplo	Resultado
=	Todos	`? n = 7`	Imprime .T. si el valor almacenado en la variable es 7; de lo contrario, imprime .F.
+	Numeric, Character,Date, DateTime	`? "Fox" + "Pro"`	Imprime “FoxPro”
! or NOT	Logical	`? !.T.`	Imprime .F. (falso)
*, /	Numeric	`? 5 * 5` `? 25 / 5`	Imprime 25 Imprime 5

Nota Un signo de interrogación (?) situado delante de una expresión imprime el resultado de la expresión y un carácter de nueva línea en la ventana de salida activa, que es normalmente la ventana principal de Visual FoxPro.

Recuerde que debe utilizar el mismo tipo de datos con cada operador. Las siguientes instrucciones almacenan dos datos numéricos en dos variables. Los nombres de variable empiezan con la letra n, por lo que se puede determinar de inmediato que contienen datos numéricos, pero puede nombrarlas con cualquier combinación de caracteres alfanuméricos y caracteres de subrayado.

```
nPrimero = 123
nSegundo = 45
```

Las instrucciones siguientes almacenan dos datos de caracteres en dos variables. Los nombres de variable empiezan con la letra c para indicar que contienen datos de tipo character.

```
cPrimero = "123"
cSegundo = "45"
```

Las dos operaciones siguientes, suma y concatenación, producen resultados distintos, ya que el tipo de datos es diferente en cada una de ellas.

```
? nPrimero + nSegundo
? cPrimero + cSegundo
```

Resultado

```
168
12345
```

Puesto que `cPrimero` contiene caracteres y `nSegundo` contiene datos numéricos, se producirá un error de tipo de datos incorrecto si se intenta ejecutar el siguiente comando:

```
? cPrimero + nSegundo
```

Puede evitar este problema si utiliza funciones de conversión. Por ejemplo, STR( ) devuelve el valor de tipo Character equivalente de un valor de tipo Numeric, mientras que VAL( ) devuelve el equivalente numérico de una cadena de caracteres formada por números. Estas funciones y LTRIM( ), que elimina los espacios iniciales, le permiten realizar las operaciones siguientes:

```
? cPrimero + LTRIM(STR(nSegundo))
? VAL(cPrimero) + nSegundo
```

Resultado

```
12345
168
```

Usar funciones

Las funciones devuelven un tipo específico de datos. Por ejemplo, las funciones STR( ) y VAL( ) utilizadas en la sección anterior devuelven valores de tipo Character y Numeric, respectivamente. Al igual que ocurre con todas las funciones, estos tipos devueltos están documentados con las funciones.

Hay cinco maneras de llamar a una función de Visual FoxPro:

- Asignar a una variable el valor que devuelve la función. La siguiente línea de código almacena la fecha actual del sistema en una variable denominada `dHoy`:

  ```
  dHoy = DATE( )
  ```

- Incluir la llamada a la función en un comando de Visual FoxPro. El siguiente comando establece el directorio predeterminado como el valor devuelto por la función GETDIR( ):

```
CD GETDIR( )
```

- Imprimir el valor devuelto en la ventana de salida activa. La siguiente línea de código imprime la hora actual del sistema en la ventana principal de Visual FoxPro:

  ```
  ? TIME( )
  ```

- Llamar a la función sin almacenar en ningún lugar el valor devuelto. La siguiente llamada de función desactiva el cursor:

  ```
  SYS(2002)
  ```

- Incluir la función dentro de otra función. La siguiente línea de código imprime el día de la semana:

  ```
  ? DOW(DATE( ))
  ```

A continuación se enumeran otros ejemplos de funciones utilizados en este capítulo:

Función	Descripción
ISDIGIT( )	Devuelve el valor verdadero (.T.) si el carácter situado al comienzo de una cadena es un número; de lo contrario, devuelve el valor falso (.F.).
FIELD( )	Devuelve el nombre de un campo.
LEN( )	Devuelve el número de caracteres de una expresión de caracteres.
RECCOUNT( )	Devuelve el número de registros de la tabla que está activa en este momento.
SUBSTR( )	Devuelve el número especificado de caracteres a partir de una cadena de caracteres, empezando en una posición especificada de la cadena.

Usar comandos

Un comando hace que se realice una determinada acción. Cada comando dispone de una sintaxis específica que indica lo que se debe incluir con el fin de que se ejecute correctamente el comando. Hay también cláusulas opcionales asociadas a los comandos que permiten especificar de forma más detallada la acción que se desea realizar.

Por ejemplo, el comando USE permite abrir y cerrar tablas:

Sintaxis de USE	Descripción
`USE`	Cierra la tabla que aparece en el área de trabajo actual.
`USE customer`	Abre la tabla CUSTOMER en el área de trabajo actual y cierra cualquier tabla que ya esté abierta

	en el área de trabajo.
`USE customer IN 0`	Abre la tabla CUSTOMER en la siguiente área de trabajo disponible.
`USE customer IN 0 ;` `  ALIAS miCliente`	Abre la tabla CUSTOMER en la siguiente área de trabajo disponible y asigna al área de trabajo el alias `miCliente`.

A continuación se muestran algunos ejemplos de comandos utilizados en este capítulo:

Comando	Descripción
DELETE	Selecciona registros especificados de una tabla para su eliminación.
REPLACE	Sustituye el valor almacenado en el campo del registro por un nuevo valor.
Go	Coloca el puntero de registro en una posición específica de la tabla.

Control del flujo del programa

Visual FoxPro incluye una categoría especial de comandos que "envuelven" a otros comandos y funciones, y determinan cuándo y con qué frecuencia se ejecutan. Estos comandos permiten realizar bifurcaciones condicionales y bucles, dos herramientas de programación muy eficaces. El siguiente programa muestra el uso de las bifurcaciones y los bucles condicionales. Estos conceptos se describen de forma más detallada después del ejemplo.

Suponga que su empresa cuenta con 10.000 empleados y desea conceder a todos aquéllos que ganan 3.000.000 de pesetas o más un aumento salarial del 3%, y a todos los que ganan menos de 3.000.000 de pesetas un aumento del 6%. El siguiente ejemplo de programa le permite hacerlo.

Este programa presupone que en el área de trabajo actual está abierta una tabla que contiene un campo numérico denominado `salario`. Si desea obtener información sobre las áreas de trabajo, consulte "Usar múltiples tablas" en el capítulo 7, Trabajar con tablas..

Programa de ejemplo para aumentar el salario de los empleados

Código	Observaciones
`SCAN`	El código comprendido entre SCAN y ENDSCAN se ejecuta tantas veces como registros haya en la tabla. Cada vez que se ejecuta el código, el puntero de registro se desplaza al siguiente registro de la tabla.

`  IF salario >= 3000000`
`  REPLACE salary WITH ;`
`  salario * 1,03`
	Para cada registro, si el salario es mayor o igual que 3.000.000, este valor se sustituye por un nuevo salario que es un 3% superior.

	El signo de punto y coma (;) que aparece después de WITH indica que el comando continúa en la siguiente línea.

`ELSE`
`  REPLACE salario WITH ;`
`  salario * 1,06`
	Para cada registro, si el salario no es mayor o igual que 3.000.000, se sustituye este valor por un nuevo salario que es un 6% superior.

`ENDIF` `ENDSCAN`	Final de la instrucción condicional IF. Final del código que se ejecuta para cada registro de la tabla.

Este ejemplo utiliza comandos de bifurcación y bucle condicional para controlar el desarrollo del programa.

Bifurcación condicional

La bifurcación condicional permite someter a prueba condiciones y, a continuación, en función del resultado de la prueba, realizar distintas operaciones. Visual FoxPro ofrece dos comandos que permiten realizar una bifurcación condicional:

- IF ... ELSE ... ENDIF
- DO CASE ... ENDCASE

El código comprendido entre la instrucción inicial y la instrucción ENDIF o ENDCASE sólo se ejecuta si una condición lógica se evalúa como verdadera (.T.). En el programa de ejemplo, el comando IF se utiliza para distinguir entre dos estados: o el salario es de 3.000.000 pesetas o más, o no lo es. Se adoptan diferentes medidas, dependiendo del estado.

En el siguiente ejemplo, si el valor almacenado en la variable `nTempAgua` es menor que 100, no se realizará ninguna acción:

```
* definir una variable lógica como Verdadera si se cumple una condición.
IF nTempAgua >= 100
   lEbullición = .T.
ENDIF
```

Nota Un asterisco al principio de una línea de un programa indica que la línea es un comentario. Los comentarios ayudan al programador a recordar la función que debe realizar cada segmento de código, si bien Visual FoxPro los pasa por alto.

Si se desea comprobar varias condiciones posibles, un bloque DO CASE ... ENDCASE puede resultar más eficaz que varias instrucciones IF y además es más fácil realizar un seguimiento del mismo.

Bucles

Un bucle le permite ejecutar una o más líneas de código tantas veces como sea necesario. En Visual

FoxPro hay tres comandos que permiten realizar bucles:

- SCAN ... ENDSCAN
- FOR ... ENDFOR
- DO WHILE ... ENDDO

Utilice SCAN cuando realice una serie de acciones para cada uno de los registros de una tabla, como en el ejemplo de programa descrito anteriormente. El bucle SCAN permite escribir el código una vez y ejecutarlo para cada registro a medida que el puntero de registro se desplaza por la tabla.

Utilice FOR cuando sepa cuántas veces debe ejecutarse la sección de código. Por ejemplo, sabe que una tabla contiene un número específico de campos. Puesto que la función FCOUNT( ) de Visual FoxPro devuelve este número, puede utilizar un bucle FOR para imprimir los nombres de todos los campos de la tabla:

```
FOR nRecuento = 1 TO FCOUNT( )
   ? FIELD(nRecuento)
ENDFOR
```

Utilice DO WHILE cuando desee ejecutar una sección de código mientras cumpla una determinada condición. Tal vez no sepa cuántas veces debe ejecutarse el código, pero sí sabe cuándo debe detenerse la ejecución. Por ejemplo, supongamos que dispone de una tabla en la que figuran los nombres y las iniciales de una serie de personas y desea utilizar las iniciales para consultar los nombres de las personas. Surgiría un problema la primera vez que intentase agregar una persona cuyas iniciales fuesen las mismas que las de otra persona contenida en la tabla.

Para resolver este problema, podría agregar un número a las iniciales. Por ejemplo, el código de identificación de Miguel Suárez podría ser MS. La siguiente persona cuyas iniciales fuesen las mismas, Margarita Sánchez, sería MS1. Si a continuación anexase María Sanz a la tabla, su código de identificación sería MS2. Un bucle DO WHILE permite localizar el número correcto que se debe adjuntar a las iniciales.

Programa de ejemplo que utiliza DO WHILE para generar un número de identificación único

Código	Comentarios
`nAquí = RECNO()`	Guardar la posición del registro.

`cIniciales = LEFT(nombre,1) + ;`
`   LEFT(apellido,1)`
`nSufijo = 0`
	Obtener las iniciales de la persona a partir de las primeras letras de los campos `nombre` y `apellido`.
	
	Si es necesario, establecer una variable que contenga el número que se debe agregar al final de las iniciales de una persona.

`LOCATE FOR id_persona = cIniciales`	Comprobar si hay otra persona en la tabla cuyas iniciales son las mismas.
`DO WHILE FOUND( )`	Si en otro registro de la tabla hay un valor id_persona que coincide con `cIniciales`, la

	función FOUND( ) devolverá el valor verdadero (.T.) y se ejecutará el código contenido en el bucle DO WHILE. Si no se encuentra ninguna coincidencia, la siguiente línea de código que se ejecute será la línea que figura a continuación de ENDDO.

`nSufijo = nSufijo + 1`
`cIniciales = ;`
`LEFT(cIniciales,2);`
`+ ALLTRIM(STR(nSufijo))`
	Preparar un sufijo nuevo y anexarlo al final de las iniciales.

`CONTINUE`	CONTINUE hace que se vuelva a evaluar el último comando LOCATE. El programa comprueba si el nuevo valor contenido en `cIniciales` ya existe en el campo id_persona de otro registro. Si es así, FOUND( ) seguirá devolviendo el valor .T. y se volverá a ejecutar el código contenido en el bucle DO WHILE. Si el nuevo valor contenido en `cIniciales` es efectivamente único, FOUND( ) devolverá el valor .F. y la ejecución del programa continuará con la línea de código que figura a continuación de ENDDO.
`ENDDO`	Final del bucle DO WHILE.
`GOTO nAquí` `REPLACE id_persona WITH cIniciales`	Volver al registro y almacenar el código de identificación único en el campo `id_persona`.

Puesto que no hay manera de saber de antemano cuántas veces se encontrarán los códigos de identificación coincidentes que ya se están utilizando, se utiliza el bucle DO WHILE.

El proceso de la programación

Cuando entienda los conceptos básicos, la programación será un proceso reiterativo. Los pasos se repiten numerosas veces, perfeccionándose el código a medida que se avanza. Al principio, someterá el código a prueba frecuentemente mediante un sistema de prueba y tanteo. Cuanto más conozca el lenguaje, mayor será la rapidez con que pueda programar y podrá realizar más pruebas preliminares mentalmente.

Entre los pasos básicos de la programación cabe citar los siguientes:

- Definir el problema.
- Desglosar el problema en elementos discretos.
- Construir los elementos.
- Comprobar y perfeccionar los elementos.
- Ensamblar los elementos.

- Comprobar el programa en su conjunto.

A continuación se enumeran algunos aspectos que debe tener presentes al empezar a programar:

- Defina claramente el problema antes de intentar resolverlo. Si no lo hace, acabará por realizar numerosos cambios, desechará código, tendrá que empezar de nuevo o bien terminará con un resultado que no es realmente lo que deseaba.
- Desglose el problema en pasos manejables, en lugar de intentar resolver todo el problema de una sola vez.
- Pruebe y depure secciones de código a medida que desarrolla el programa. Compruebe que el código hace lo que quiere que haga. La depuración es el proceso de encontrar y solucionar problemas que impiden que el código se ejecute correctamente.
- Perfeccione los datos y el almacenamiento de datos para facilitar la manipulación de estos datos a través del código del programa. Esto suele implicar estructurar las tablas de forma adecuada.

En el resto de esta sección se describen los pasos que debe seguir para construir un pequeño programa Visual FoxPro.

Definir el problema

Antes de poder resolver un problema, debe formularlo claramente. Algunas veces, si ajusta la formulación del problema podrá ver más opciones para resolverlo.

Suponga que obtiene muchos datos de distintos orígenes. Si bien la mayoría de los datos son estrictamente numéricos, algunos valores contienen guiones y espacios en blanco además de números. Suponga que quiere eliminar todos los espacios en blanco y los guiones de dichos campos y guardar los datos numéricos.

En lugar de intentar eliminar los espacios en blanco y los guiones de los datos originales, podría formular el objetivo del programa como:

Objetivo Reemplazar los valores existentes de un campo por otros valores que contengan todo lo que contenían los valores originales, excepto los espacios en blanco y los guiones.

Esta formulación evita la dificultad que supone manipular una cadena de caracteres cuya longitud sigue cambiando a medida que trabaja con ella.

Descomponer el problema

Puesto que tiene que indicar instrucciones específicas a Visual FoxPro en términos de operaciones, comandos y funciones, debe descomponer el problema en pasos discretos. La tarea más discreta para este problema sería examinar cada carácter de la cadena. Hasta que pueda examinar un carácter individualmente, no podrá determinar si desea guardarlo.

Una vez que examine un carácter, deberá comprobar si se trata de un guión o de un espacio en blanco. En este momento, quizá desee refinar la declaración del problema. ¿Y si obtuviera más adelante datos que contienen paréntesis de apertura y de cierre? ¿Y si desea deshacerse de los símbolos de moneda, las comas y los puntos? Cuanto más genérico pueda hacer el código, más

trabajo se ahorrará de ahora en adelante; lo principal es ahorrar trabajo. He aquí una formulación del problema válida para una variedad mucho mayor de datos:

Objetivo refinado Reemplazar los valores existentes en un campo por otros valores que contengan únicamente los caracteres numéricos de los valores originales.

Con esta formulación, ahora puede volver a plantear el problema a nivel de carácter: si el carácter es numérico, guardar el carácter; si el carácter es no numérico, pasar al siguiente carácter. Cuando haya construido una cadena que sólo contenga los elementos numéricos de la cadena inicial, podrá reemplazar la primera cadena y pasar al siguiente registro hasta que haya terminado con todos los datos.

Para resumir, el problema se descompone en los siguientes elementos:

1. Examinar cada carácter.
2. Decidir si el carácter es numérico o no.
3. Si es numérico, copiarlo a la segunda cadena.
4. Cuando haya terminado con todos los caracteres de la cadena original, reemplazar la cadena original con la cadena que sólo contiene valores numéricos.
5. Repetir estos pasos para todos los registros de la tabla.

Construir los elementos

Cuando sepa qué debe hacer, puede empezar a formular los elementos en términos de comandos, funciones y operadores de Visual FoxPro.

Como los comandos y funciones se usan para manipular datos, tiene algunos datos de prueba para trabajar con ellos. Los datos de prueba sirven para simular los datos verdaderos lo mejor posible.

Para este ejemplo puede almacenar en una variable una cadena de prueba introduciendo el siguiente comando en la ventana Comandos:

```
cTest = "123-456-7 89 0"
```

Examinar cada carácter

En primer lugar desea buscar un único carácter en la cadena. Para obtener una lista de funciones que se pueden utilizar para manipular cadenas, vea Funciones de carácter.

Verá tres funciones que devuelven determinadas secciones de una cadena: LEFT( ), RIGHT( ) y SUBSTR( ). De estas tres funciones, SUBSTR( ) devuelve caracteres de cualquier parte de la cadena.

SUBSTR( ) usa tres argumentos o parámetros: la cadena, la ubicación inicial dentro de la cadena y el número de caracteres que se deben devolver de la cadena, empezando por la ubicación inicial. Para comprobar si SUBSTR( ) va a hacer lo que usted quiere, podría escribir los siguientes comandos en la

ventana Comandos:

```
? SUBSTR(cTest, 1, 1)
? SUBSTR(cTest, 3, 1)
? SUBSTR(cTest, 8, 1)
```

Resultado

```
1
3
-
```

Puede ver que en la ventana principal de Visual FoxPro se muestran el primer, el tercer y el octavo caracteres de la cadena de prueba.

Para hacer eso mismo varias veces, utilice un bucle. Puesto que la cadena de prueba tiene un número determinado de caracteres (14), puede utilizar un bucle FOR. El contador del bucle FOR se incrementa cada vez que se ejecuta el código del bucle, por lo que puede utilizar el contador de la función SUBSTR( ). Puesto que en la ventana Comandos no puede comprobar las construcciones de bucles, deberá probar la siguiente sección de código en un programa de ejemplo.

Para crear un programa nuevo

1. Escriba el siguiente comando en la ventana **Comandos**:

   ```
   MODIFY COMMAND numonly
   ```

2. En la ventana que aparecerá, escriba las siguientes líneas de código:

   ```
   FOR nCnt = 1 TO 14
      ? SUBSTR(cTest, nCnt, 1)
   ENDFOR
   ```

Ahora que ha creado un programa, ya puede ejecutarlo.

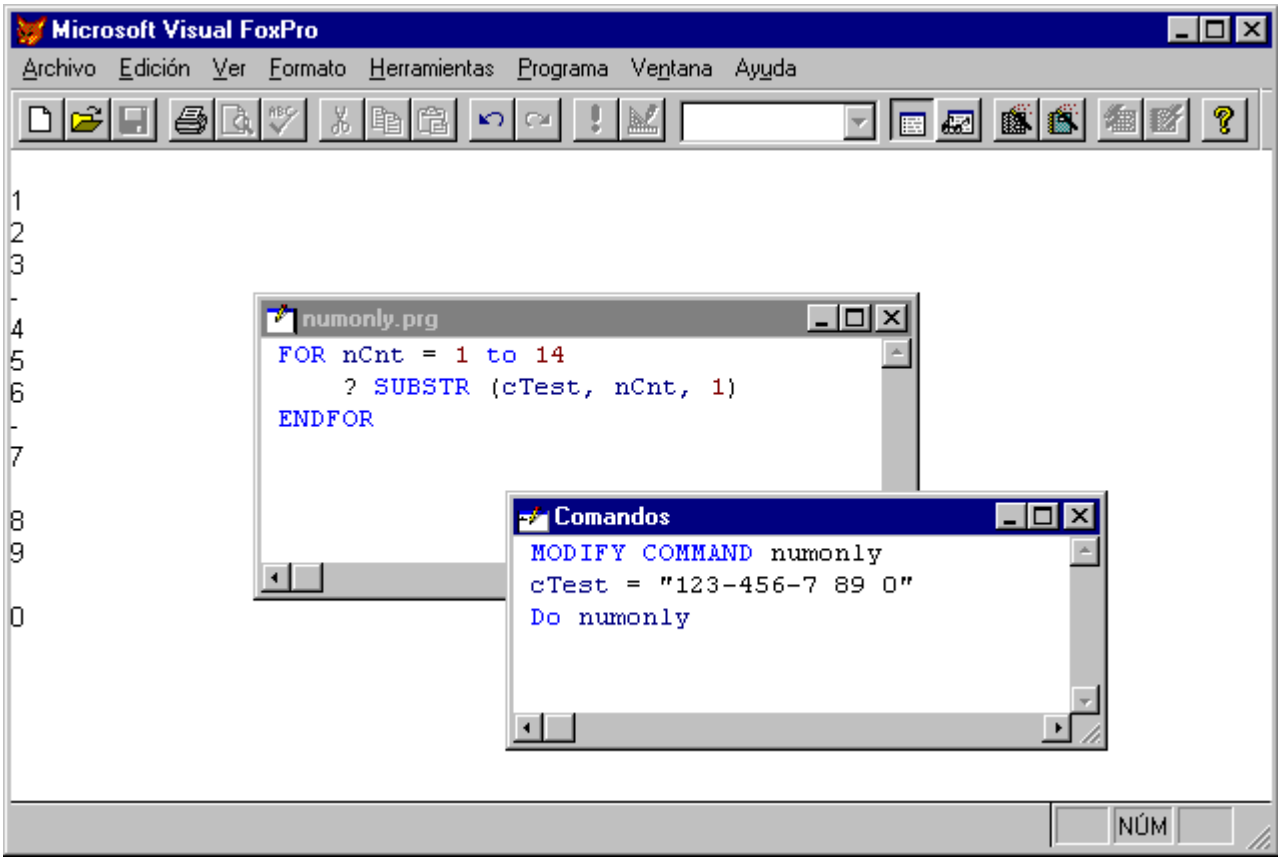
Para ejecutar un programa

1. En la ventana del programa abierto, presione CTRL+E.

2. Si aparece un cuadro de diálogo **Guardar**, elija **Aceptar**.

Cuando ejecute este programa, los caracteres individuales de la cadena de prueba se imprimirán en líneas distintas en la ventana principal de Visual FoxPro.

Comprobar parte del programa



Ya ha completado la primera tarea. Ahora puede examinar cada carácter de la cadena.

Decidir si el carácter es numérico

Cuando tenga un único carácter de la cadena, debe saber si se trata de un número. Puede hacerlo con ISDIGIT( ).

Puede probar los siguientes comandos en la ventana Comandos:

```
? ISDIGIT('2')
? ISDIGIT('-')
? ISDIGIT(SUBSTR(cTest, 3, 1))
```

Resultado

```
.T.
.F.
.T.
```

De este resultado se desprende que ‘2’ es un número, ‘ – ’ no es un número y el tercer carácter, 3, es un número.

Si el carácter es numérico, copiarlo a la segunda cadena

Ahora que puede examinar los caracteres y determinar si son o no numéricos, necesita una variable

para almacenar los valores numéricos: `cNumOnly`.

Para crear la variable, debe asignarle un valor inicial, una cadena de longitud cero:

```
cNumOnly = ""
```

A medida que el bucle FOR recorre la cadena, es conveniente crear otra variable para almacenar temporalmente cada carácter de la cadena a medida que ésta se manipula:

```
cCharacter = SUBSTR(cTest, nCnt, 1)
```

Sugerencia Normalmente es mejor almacenar en una variable de memoria el resultado de un cálculo, evaluación o función. Entonces puede manipular la variable en lugar de tener que repetir el cálculo o la evaluación.

La siguiente línea de código puede utilizarse cada vez que se encuentra un número para sumar el número a la segunda cadena:

```
cNumOnly = cNumOnly + cCharacter
```

Hasta ahora, el programa es el siguiente:

```
cNumOnly = ""
FOR nCnt = 1 TO 14
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
```

Prueba de los elementos

Si agrega un par de comandos al final para imprimir las cadenas y ejecutar el programa, podrá ver que el programa funciona con la cadena de prueba:

```
cNumOnly = ""
FOR nCnt = 1 TO 14
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? cTest
? cNumOnly
```

Resultado

```
123-456-7 89 0
1234567890
```

El resultado parece correcto. Pero si cambia la cadena de prueba mientras comprueba los elementos, puede tener problemas. Escriba el siguiente comando en la ventana Comandos y ejecute de nuevo el programa:

```
cTest = "456-789 22"
```

El programa generará un mensaje de error. El bucle FOR ha intentado ejecutarse 14 veces, pero en la cadena sólo había 10 caracteres. Necesita una forma de ajustar las longitudes variables de las cadenas. Use LEN( ) para devolver el número de caracteres de una cadena. Si sustituye este comando en el bucle FOR, verá que el programa funciona correctamente con ambas cadenas de prueba:

```
cNumOnly = ""
FOR nCnt = 1 TO LEN(cTest)
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? cTest
? cNumOnly
```

Agrupar los elementos

Para completar la solución de programación para este problema, quizá desee volver a leer sus datos de una tabla. Cuando tenga una tabla, explore los registros y aplique su código de programa a un campo de la tabla, en lugar de a una variable.

En primer lugar, podría crear una tabla temporal que contuviera diversas cadenas de prueba. Dicha tabla podría contener un único campo de caracteres llamado Testfield y cuatro o cinco registros:

Contenido de Testfield	
123-456-7 89 0	-9221 9220 94321 99-
456-789 22	000001 98-99-234

Cuando sustituya el nombre de la cadena de prueba por el nombre del campo, el programa será similar al siguiente:

```
FOR nCnt = 1 TO LEN(TestField)
   cCharacter = SUBSTR(TestField, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? TestField
? cNumOnly
```

Puede ajustar manualmente el puntero de registro si examina la tabla y se desplaza por ella. Cuando el puntero de registro esté en cada uno de los registros, el programa funcionará de la forma deseada. O bien, ahora puede envolver el código de desplazamiento por la tabla en el resto de su programa:

```
SCAN
   cNumOnly = ""
   FOR nCnt = 1 TO LEN(TestField)
      cCharacter = SUBSTR(TestField, nCnt, 1)
      IF ISDIGIT(cCharacter)
```

```
         cNumOnly = cNumOnly + cCharacter
      ENDIF
   ENDFOR
? TestField
? cNumOnly
?
ENDSCAN
```

Resultado

```
123-456-7 89 0
1234567890

456-789 22
45678922

-9221 9220 94321 99-
922192209432199

000001 98-99-234
0000019899234
```

Comprobar todo el programa

En lugar de imprimir la cadena al final del programa, quizá desee guardarla en la tabla. Para ello, utilice la siguiente línea de código:

```
REPLACE TestField WITH cNumOnly
```

El programa completo será el siguiente:

```
SCAN
   cNumOnly = ""
   FOR nCnt = 1 TO LEN(TestField)
      cCharacter = SUBSTR(TestField, nCnt, 1)
      IF ISDIGIT(cCharacter)
         cNumOnly = cNumOnly + cCharacter
      ENDIF
   ENDFOR
   REPLACE TestField WITH cNumOnly
ENDSCAN
```

Cuando tenga el programa completo, necesitará probarlo con los datos de ejemplo antes de probarlo con los datos reales.

Aumentar la robustez del programa

Un programa robusto hace lo que usted quiere que haga, pero también se anticipa a posibles problemas y se encarga de ellos. Este programa hace lo que usted quiere, pero hace algunas suposiciones que deben ser verdaderas para que funcione:

- Hay una tabla abierta en el área de trabajo actual.
- La tabla tiene un campo de caracteres llamado `TestField`.

Si la tabla no está abierta en el área de trabajo actual o si la tabla no tiene un campo de caracteres con

el nombre esperado, el programa generará un mensaje de error y no realizará la tarea prevista.

Programa para eliminar los caracteres no numéricos de todos los registros de un campo

Código	Comentarios
`lFieldOK = .F.`	Esta variable determina si existen las condiciones necesarias para que el programa funcione. Inicialmente se establece la variable como falsa (.F.) para suponer que las condiciones necesarias no existen.

`FOR nCnt = 1 TO FCOUNT( )`
`  IF FIELD(nCnt) = ;`
`  UPPER("TestField")`
`  IF TYPE("TestField") = "C"`
`  lFieldOK = .T.`
`  ENDIF`
`  EXIT`
`  ENDIF`
`ENDFOR`
	Esta sección de código recorre todos los campos de la tabla actual hasta que encuentra un campo de caracteres llamado `TestField`. En cuanto se encuentra el campo correcto, `lFieldOK` se establece como verdadera (.T.) y EXIT finaliza el bucle (no hay ninguna razón para continuar con la comprobación una vez identificado el campo correcto). Si ningún campo cumple el criterio, `lFieldOK` seguirá siendo falso (.F.).

`IF lFieldOK`	La sección de conversión del programa sólo se ejecuta si en la tabla activa actualmente hay un campo de caracteres llamado `TestField`.

`SCAN`
`  cNumOnly = ""`
`  FOR nCnt = 1 TO LEN(TestField)`
`  cCharacter = ;`
`  SUBSTR(TestField, nCnt, 1)`
`  IF ISDIGIT(cCharacter)`
`  cNumOnly = cNumOnly + ;`
`  cCharacter`
`  ENDIF`
`  ENDFOR`
	El código de conversión.

`  REPLACE TestField WITH ;`
`  cNumOnly`
`ENDSCAN`	

`ENDIF`	Fin de la condición `IF lFieldOK`.

La mayor limitación de este programa es que sólo puede utilizarlo para un campo. Si desea eliminar los caracteres no numéricos de un campo distinto de `TestField`, tendrá que recorrer el programa y cambiar cada aparición de `TestField` por el nombre del otro campo.

Convertir el programa a un procedimiento, según se explica en las próximas secciones, permite hacer que el código que ha escrito sea más genérico y más reutilizable, con lo que ahorrará trabajo más adelante.

Usar procedimientos y funciones definidas por el usuario

Los procedimientos y funciones permiten mantener en un único lugar el código que utiliza con frecuencia y llamarlo a través de su aplicación siempre que lo necesite. Esto hace que su código sea más fácil de leer y mantener, ya que en un procedimiento el cambio se realiza una sola vez, no varias veces como ocurre en un programa.

En Visual FoxPro, los procedimientos son similares a éste:

```
PROCEDURE miproc
  * Esto es un comentario, pero podría ser código ejecutable
ENDPROC
```

Tradicionalmente, los procedimientos contienen código que usted escribe para realizar una operación y funciones que calculan y devuelven un valor. En Visual FoxPro, las funciones son similares a los procedimientos:

```
FUNCTION mifunción
  * Esto es un comentario, pero podría ser código ejecutable
ENDFUNC
```

Puede incluir procedimientos y funciones en un archivo de programa distinto o al final de un archivo de programa que contenga código normal de programa. En un archivo de programa no puede tener código ejecutable de programa a continuación de los procedimientos y las funciones.

Si incluye sus procedimientos y funciones en un archivo de programa distinto, podrá hacer accesibles estos procedimientos y funciones desde su programa si utiliza el comando SET PROCEDURE TO. Por ejemplo, para un archivo llamado FUNPROC.PRG, utilice el siguiente comando en la ventana Comandos:

```
SET PROCEDURE TO funproc.prg
```

Llamar a un procedimiento o a una función

Hay dos formas de llamar a un procedimiento o a una función en sus programas:

- Utilizar el comando DO. Por ejemplo:

  ```
  DO miproc
  ```

  –O bien–

- Incluir detrás del nombre de la función un par de paréntesis. Por ejemplo:

  ```
  mifunción( )
  ```

Cada uno de estos métodos puede ampliarse enviando o recibiendo valores desde el procedimiento o la función.

Enviar valores a un procedimiento o a una función

Para enviar valores a procedimientos o funciones se incluyen parámetros. Por ejemplo, el procedimiento siguiente acepta un solo parámetro:

```
PROCEDURE miproc( cString )
   * La línea siguiente muestra un mensaje
   MESSAGEBOX ("miproc" + cString)
ENDPROC
```

Nota Incluir los parámetros dentro de los paréntesis en la línea de definición de un procedimiento o una función, por ejemplo `PROCEDURE miproc(cString)`, indica que el parámetro tiene alcance local al procedimiento o la función. También puede permitir que una función o un procedimiento acepte parámetros de alcance local mediante LPARAMETERS.

Los parámetros funcionan de manera idéntica en una función. Para enviar un valor como un parámetro de este procedimiento o a una función, podría utilizar una cadena o una variable que contuviera una cadena, como se muestra en la tabla siguiente.

Transferencia de parámetros

Código	Comentarios
`DO miproc WITH cTestString` `DO miproc WITH "cadena de prueba"`	Llama a un procedimiento y transfiere una variable de caracteres o un literal de cadena.
`mifunción("cadena de prueba")` `mifunción( cTestString )`	Llama a una función y transfiere una copia de una cadena literal o una variable de caracteres.

Nota Si llama un procedimiento o función sin usar el comando DO, la configuración de UDFPARMS controla cómo se transfieren los parámetros. De forma predeterminada, UDFPARMS se establece como VALUE, por lo que se transferirán copias de los parámetros. Cuando utilice DO, se empleará el parámetro real (el parámetro se transfiere por referencia) y cualquier cambio realizado en el procedimiento o en la función se reflejarán en los datos originales, cualquiera que sea la configuración de UDFPARMS.

Puede enviar múltiples valores a un procedimiento o función si los separa mediante comas. Por ejemplo, el siguiente procedimiento espera tres parámetros: una fecha, una cadena de caracteres y un número.

```
PROCEDURE miproc( dDate, cString, nTimesToPrint )
   FOR nCnt = 1 to nTimesToPrint
      ? DTOC(dDate) + " " + cString + " " + STR(nCnt)
   ENDFOR
ENDPROC
```

Podría llamar a este procedimiento mediante la siguiente línea de código:

```
DO miproc WITH DATE(), "Hola", 10
```

Recibir valores desde una función

El valor devuelto de forma predeterminada es verdadero (.T.), pero puede utilizar el comando RETURN para devolver cualquier valor. Por ejemplo, la siguiente función devuelve una fecha correspondiente a dos semanas después de la fecha que se ha pasado como parámetro.

```
FUNCTION plus2weeks
PARAMETERS dDate
   RETURN dDate + 14
ENDFUNC
```

La siguiente línea de código almacena el valor devuelto desde esta función en una variable:

```
dDeadLine = plus2weeks(DATE())
```

En la tabla siguiente se muestran las formas en que puede almacenar o mostrar valores devueltos por una función.

Manipular valores devueltos

Código	Comentarios
`var = mifunción( )`	Almacena en una variable el valor devuelto por la función.
`? mifunción( )`	Imprime en la ventana activa de salida el valor devuelto por la función.

Comprobar parámetros en un procedimiento o en una función

Es conveniente comprobar que los parámetros enviados a su procedimiento o a su función son los que espera recibir. Puede utilizar las funciones TYPE( ) y PARAMETERS( ) para comprobar el tipo y el número de parámetros enviados a su procedimiento o a su función.

El ejemplo de la sección anterior necesita recibir un parámetro de tipo Fecha. Puede utilizar la función TYPE( ) para asegurarse de que el valor que su función recibe es del tipo adecuado.

```
FUNCTION plus2weeks( dDate )
   IF TYPE("dDate") = "D"
      RETURN dDate + 14
   ELSE
      MESSAGEBOX( "requiere un parámetro de tipo Fecha" )
      RETURN {}      && Devuelve una fecha vacía
   ENDIF
ENDFUNC
```

Si un procedimiento espera menos parámetros de los que recibe, Visual FoxPro generará un mensaje de error. Por ejemplo, si especificó dos parámetros pero llamó al procedimiento con tres parámetros, obtendrá un mensaje de error. Pero si un procedimiento espera más parámetros de los que recibe, los parámetros adicionales simplemente se inicializarán como falso (.F.). Puesto que no hay ninguna forma de decir si el último parámetro se estableció como falso (.F.) o se omitió, el siguiente

procedimiento comprueba que se ha enviado el número correcto de parámetros:

```
PROCEDURE SaveValue( cStoreTo, cNewVal, lIsInTable )
   IF PARAMETERS( ) < 3
      MESSAGEBOX( "No se han pasado suficientes parámetros". )
      RETURN .F.
   ENDIF
   IF lIsInTable
      REPLACE (cStoreTo) WITH (cNewVal)
   ELSE
      &cStoreTo = cNewVal
   ENDIF
   RETURN .T.
ENDPROC
```

Convertir el programa NUMONLY en una función

NUMONLY.PRG, el programa de ejemplo descrito en la sección El proceso de la programación, puede hacerse más robusto y útil si crea una función para la parte del programa que elimina los caracteres no numéricos de una cadena.

Procedimiento de ejemplo para devolver caracteres numéricos de una cadena

Código	Comentarios
`FUNCTION NumbersOnly( cMixedVal )`	Principio de la función, que acepta una cadena de caracteres.

`cNumOnly = ""`
`FOR nCnt = 1 TO LEN(cMixedVal)`
`cCharacter = ;`
`SUBSTR(cMixedVal, nCnt, 1)`
`IF ISDIGIT(cCharacter)`
`cNumOnly = ;`
`cNumOnly + cCharacter`
`ENDIF`
`ENDFOR`
	Crea una cadena que sólo tiene los caracteres numéricos de la cadena original.

`RETURN cNumOnly`	Devuelve la cadena que sólo tiene caracteres numéricos.
`ENDFUNC`	Fin de la función.

Además de permitirle utilizar este código en múltiples situaciones, esta función mejora la legibilidad del programa:

```
SCAN
   REPLACE FieldName WITH NumbersOnly(FieldName)
ENDSCAN
```

O, más sencillo aún:

```
REPLACE ALL FieldName WITH NumbersOnly(FieldName)
```

Cómo avanzar

La programación por procedimientos, junto con la programación orientada a objetos y las herramientas de diseño de Visual FoxPro, pueden ayudarle a programar una versátil aplicación de Visual FoxPro. El resto de este manual explica temas con los que se encontrará a medida que programe aplicaciones de Visual FoxPro.

Para obtener más información acerca de la programación orientada a objetos, consulte el capítulo 3, Programación orientada a objetos. Para aprender a diseñar formularios con el Diseñador de formularios, consulte el capítulo 9,Crear formularios.

Capítulo 2: Programar una aplicación

Una aplicación de Visual FoxPro incluye normalmente una o más bases de datos, un programa principal que configura el entorno del sistema para la aplicación y una interfaz de usuario compuesta por formularios, barras de herramientas y menús. Las consultas y los informes permiten que los usuarios extraigan información de sus datos.

Este capítulo trata los temas siguientes:

- Diseñar la aplicación
- Crear bases de datos
- Crear clases
- Proporcionar acceso a la funcionalidad
- Proporcionar acceso a la información
- Pruebas y depuración

- Para obtener información acerca de la programación de aplicaciones con el Generador de aplicaciones y el Marco de aplicaciones mejorado, vea Programar aplicaciones con el Marco de aplicaciones.

Diseñar la aplicación

Un diseño apropiado ahorra tiempo, esfuerzo, dinero y permite mantener la cordura al programar. Cuanto más implique a los usuarios en el proceso de diseño, mejor. No importa lo cuidadosamente que se diseñe; aun así, acabará refinando las especificaciones a medida que avance el proyecto y los usuarios le proporcionen información adicional.

Algunas de las decisiones de diseño que tome afectarán a la forma en que creará elementos de la aplicación. ¿Quién utilizará la aplicación? ¿Cuál es el centro de actividad del usuario? ¿Con qué cantidad de datos se supone que se trabajará? ¿Se utilizarán servidores de datos de apoyo o los datos serán exclusivamente locales para un único usuario o para múltiples usuarios a través de una red? Considere estos factores antes de avanzar demasiado en el proyecto.

Actividades frecuentes de usuario

Incluso si sus usuarios finales trabajan con clientes, pedidos y piezas, lo que determinará la forma en que la aplicación deberá tratar los datos es el modo en que los usuarios trabajan con esa información. Un formulario de entrada de pedidos, como el de Tastrade.app (en el directorio ...\Samples\Vfp98 \Tastrade de Visual Studio), puede ser necesario para algunas aplicaciones, pero no sería una buena herramienta para administrar inventarios o para hacer un seguimiento de las ventas, por ejemplo.

Tamaño de la base de datos

Deberá pensar más en el rendimiento si trata con grandes conjuntos de datos. En el capítulo 15, Optimizar aplicaciones, se explican los métodos para optimizar el rendimiento. También puede desear ajustar el modo en que los usuarios pueden desplazarse por los datos. Si tiene veinte o treinta registros en una tabla, está bien que permita que los usuarios desplacen el puntero de registro de una tabla un registro cada vez. Si tiene veinte o treinta mil registros, deberá proporcionar otros sistemas para obtener los datos deseados: agregar listas o cuadros de diálogos de búsqueda, filtros, consultas personalizadas, etc. El capítulo 10, Usar controles, explica la forma de utilizar una lista para seleccionar registros específicos de una tabla. En el capítulo 8, Crear vistas, se explica la creación de consultas parametrizadas.

Usuario individual frente a múltiples usuarios

Es conveniente crear la aplicación pensando que múltiples usuarios tendrán acceso simultáneo a la base de datos. Visual FoxPro facilita la programación para acceso compartido. En el capítulo 17, Programar para acceso compartido, se describen técnicas para permitir que varios usuarios tengan acceso simultáneo a la base de datos.

Consideraciones internacionales

Si sabe que su aplicación sólo se utilizará en el entorno de un único idioma, no debe preocuparse de la internacionalización. Por otra parte, si desea ampliar su mercado, o si sus usuarios deben trabajar con datos o configuraciones de entorno internacionales, deberá tener en cuenta estos factores al crear la aplicación. En el capítulo 18, Programar aplicaciones internacionales, se explican los puntos que debe tener en cuenta cuando programe aplicaciones internacionales.

Datos locales frente a datos remotos

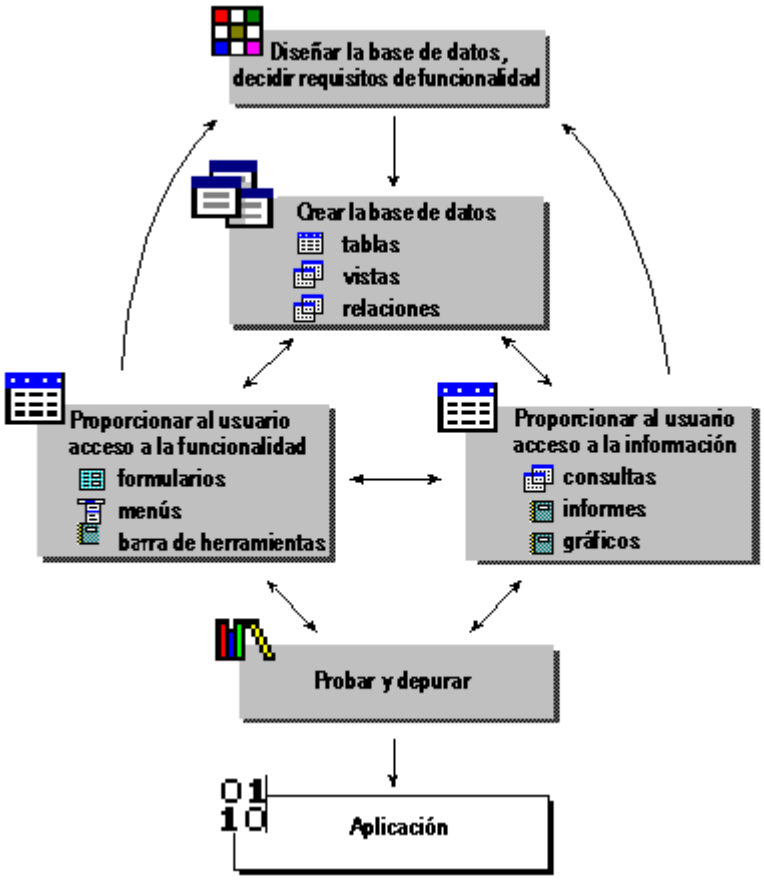
Si su aplicación trata con datos remotos, deberá administrarlos de forma diferente a como administraría los datos nativos de Visual FoxPro. En el capítulo 8, Crear vistas, se explica la forma de crear vistas para datos locales o remotos. En la parte 6 del *Manual del programador*, Crear soluciones cliente-servidor, se explica cómo diseñar aplicaciones que trabajen sin problemas con datos remotos.

Descripción general del proceso

El proceso de crear una aplicación es muy repetitivo. Puesto que no hay dos aplicaciones exactamente iguales, lo que hará probablemente será definir prototipos y refinar algunos componentes varias veces antes de obtener un producto final. Las expectativas de los usuarios finales, o sus solicitudes, también pueden cambiar, lo que hará necesario redefinir aspectos de la aplicación. Además, nadie escribe código libre de errores, por lo que las pruebas y la depuración conducen a algún tipo de rediseño o

rescritura.

El proceso de creación de una aplicación



Además de tener en cuenta la imagen general durante la fase de diseño, tendrán que decidir cuáles son las funciones necesarias, qué datos están implicados y cómo debe estar estructurada la base de datos. Tendrá que diseñar una interfaz para que el usuario tenga acceso a la funcionalidad de la aplicación. Puede crear informes y consultas para que los usuarios puedan extraer información útil de sus datos.

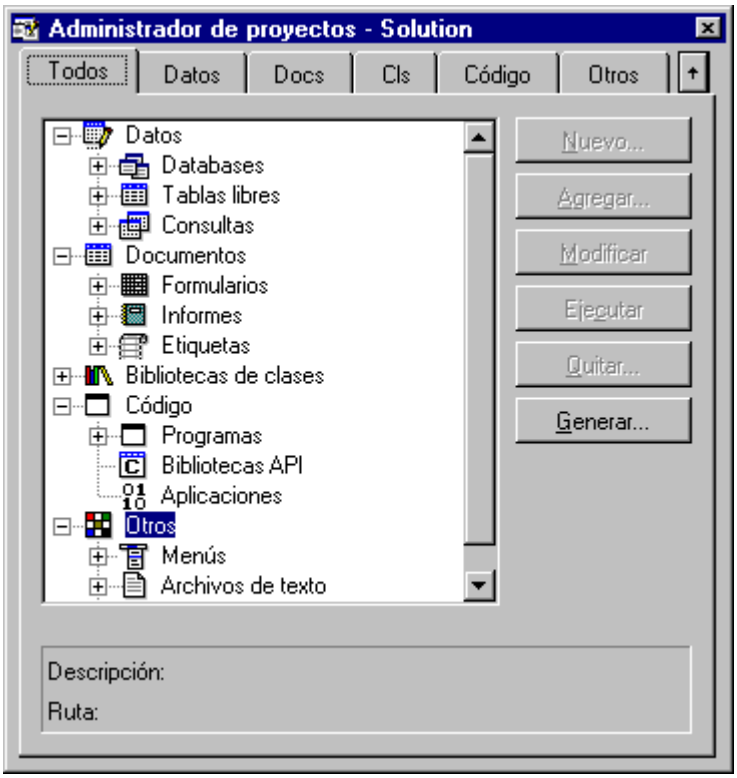
Iniciar la programación

Después de haber pensado qué componentes son necesarios en la aplicación, es posible que quiera configurar un conjunto de directorios y un proyecto para organizar los archivos componentes que desea crear para la aplicación. Puede crear el conjunto de directorios mediante el Explorador de Windows y puede crear el proyecto en el Administrador de proyectos o usar el Asistente para aplicaciones para configurarlos a la vez. Este nuevo Asistente para aplicaciones abre el Generador de aplicaciones para que pueda personalizar aún más un proyecto y los componentes que inicie en el Asistente. Por compatibilidad con versiones anteriores, puede elegir el Asistente para aplicaciones (5.0) anterior.

Usar el Administrador de proyectos

El Administrador de proyectos permite compilar la aplicación completa. En la fase de programación de la aplicación, el Administrador de proyectos facilita el diseño, la modificación y la ejecución de los componentes individuales de su aplicación.

El Administrador de proyectos



Cuando utilice el Administrador de proyectos, podrá:

- Modificar y ejecutar partes de su aplicación (formularios, menús, programas) con tan sólo algunos clics.
- Arrastrar clases, tablas y campos desde el Administrador de proyectos hasta el Diseñador de formularios o el Diseñador de clases.
- Arrastrar clases entre bibliotecas de clases.
- Ver y modificar fácilmente sus tablas y bases de datos.
- Agregar descripciones para los componentes de la aplicación.
- Arrastrar y colocar elementos entre proyectos.

Para obtener información detallada acerca del uso del Administrador de proyectos, consulte el capítulo 1, Introducción, del *Manual del usuario*. Para obtener información acerca de la compilación de aplicaciones, consulte el capítulo 13, Compilar una aplicación, en este manual.

Crear bases de datos

Puesto que una aplicación de base de datos depende tanto de los datos subyacentes, la mejor forma de empezar a diseñar su aplicación es comenzar por los datos. Puede configurar su propia base de datos y determinar cuáles serán las relaciones entre las tablas, qué reglas comerciales va a exigir, etc. antes

de diseñar cualquier componente de interfaz o de manipulación de datos. La creación de una estructura sólida de la base de datos hará que el trabajo de programación sea mucho más sencillo.

En el capítulo 5, Diseñar bases de datos, el capítulo 6, Crear bases de datos y el capítulo 7, Trabajar con tablas, se explican los temas de diseño y uso de Visual FoxPro para diseñar tablas y bases de datos efectivas y eficaces.

Crear clases

Puede crear una aplicación robusta, orientada a objetos y controlada por eventos utilizando únicamente las clases de base de Visual FoxPro. Es posible que no tenga que crear una clase, pero deseará hacerlo. Además de hacer que el código sea más manejable y sencillo de mantener, una biblioteca de clases sólida le permite crear rápidamente prototipos e incorporar funciones a una aplicación. Puede crear clases en un archivo de programa, en el Diseñador de formularios (mediante el comando **Guardar como clase** del menú **Archivo**) o en el Diseñador de clases.

El capítulo 3, Programación orientada a objetos, trata algunos de los beneficios de la creación de clases y detalla su creación con el Diseñador de clases o mediante programación.

Proporcionar acceso a la funcionalidad

La satisfacción del usuario se verá influenciada en gran medida por la interfaz que le proporcione para la funcionalidad de su aplicación. Puede tener un modelo de clases muy limpio, código muy elegante y soluciones muy inteligentes para los problemas difíciles de su aplicación, pero esto casi siempre está oculto a los clientes. Lo que ellos ven es la interfaz que usted les proporciona. Afortunadamente, las herramientas de diseño de Visual FoxPro facilitan la creación de interfaces atractivas y ricas en características.

La interfaz de usuario consiste principalmente en formularios, barras de herramientas y menús. Puede asociar toda la funcionalidad de su aplicación con controles o comandos de menú en la interfaz. En el capítulo 9, Crear formularios, se describe la creación de formularios y conjuntos de formularios. La utilización de controles de Visual FoxPro en los formularios se trata en el capítulo 10, Usar controles. Consulte el capítulo 11, Diseñar menús y barras de herramientas, para dar los últimos retoques a la aplicación.

Proporcionar acceso a la información

Probablemente muestre cierta información para los usuarios en formularios, pero también deseará ofrecer a los usuarios la posibilidad de especificar exactamente la información que desean ver, así como la opción de imprimirla en informes o etiquetas. Las consultas, especialmente las consultas que aceptan parámetros definidos por el usuario, permiten a los usuarios tener más control sobre sus datos. Los informes permiten a los usuarios imprimir imágenes totales, parciales o de resumen de sus datos. Los controles ActiveX y la automatización permiten que su aplicación comparta información y funciones con otras aplicaciones.

El Diseñador de consultas y el Diseñador de informes se describen en los capítulos 4 a 7 del *Manual del usuario*. En el capítulo 12 de este manual, Agregar consultas e informes, se explica la integración de consultas e informes en una aplicación. El capítulo 16, Agregar OLE describe la integración de

OLE en una aplicación.

Probar y depurar

La prueba y depuración es algo que la mayoría de los programadores hace en cada paso del proceso de desarrollo. Es conveniente probar y depurar a medida se va avanzando. Si crea un formulario, querrá asegurarse de que éste hace lo que se desea antes de continuar con otros elementos de su aplicación.

En el capítulo 14, Probar y depurar aplicaciones, se explica el uso de las herramientas de depuración de Visual FoxPro para depurar sus aplicaciones y se ofrecen sugerencias para que el proceso resulte más sencillo.

Capítulo 3: Programación orientada a objetos

Aunque Visual FoxPro admite la programación estándar por procedimientos, se ha ampliado la capacidad del lenguaje para proporcionar la potencia y la flexibilidad propias de la programación orientada a objetos.

El diseño orientado a objetos y la programación orientada a objetos representan un cambio de perspectiva con respecto a la programación estándar por procedimientos. En lugar de pensar en el flujo del programa desde la primera hasta la última línea de código, se debe pensar en la creación de objetos: componentes autocontenidos de una aplicación que tienen funcionalidad privada además de la funcionalidad que se puede exponer al usuario.

En este capítulo se tratan los temas siguientes:

- Descripción de los objetos de Visual FoxPro
- Descripción de las clases de Visual FoxPro
- Adaptar la clase a la tarea
- Crear clases
- Agregar clases a formularios
- Definir clases mediante programación

Descripción de los objetos de Visual FoxPro

En Visual FoxPro, los formularios y los controles son objetos que puede incluir en sus aplicaciones. Puede manipular estos objetos a través de sus propiedades, eventos y métodos.

Las mejoras en el lenguaje orientado a objetos de Visual FoxPro proporcionan un mayor control sobre los objetos de las aplicaciones. Asimismo, facilitan la creación y el mantenimiento de bibliotecas de código reutilizable, proporcionando:

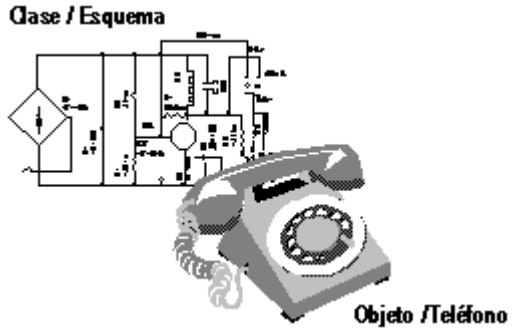
- Código más compacto.
- Incorporación más sencilla del código a las aplicaciones sin necesidad de elaborar esquemas de asignación de nombres.
- Menos complejidad al integrar código de distintos archivos en una aplicación.

La programación orientada a objetos es en gran medida un modo de empaquetar código de manera que se pueda volver a utilizar y mantener más fácilmente. Los paquetes principales se llaman clases.

Clases y objetos: los bloques funcionales de las aplicaciones

Las clases y los objetos están estrechamente relacionados, pero no son lo mismo. Una clase contiene información sobre cuál debe ser la apariencia y el comportamiento de un objeto. Una clase es el plano o esquema de un objeto. Por ejemplo, el esquema eléctrico y de diseño de un teléfono sería algo similar a una clase. El objeto o una instancia de la clase sería el teléfono.

La clase determina las características del objeto.



Los objetos tienen propiedades

Un objeto tiene ciertas propiedades o atributos. Por ejemplo, un teléfono tiene un color y un tamaño determinados. Cuando se instala un teléfono en la oficina, tiene una determinada posición sobre la mesa. El receptor puede estar colgado o descolgado.

Los objetos que se crean en Visual FoxPro también tienen propiedades que están determinadas por la clase en la que se basa el objeto. Estas propiedades pueden establecerse en tiempo de diseño o en tiempo de ejecución.

Por ejemplo, en la tabla siguiente se indican algunas propiedades que puede tener una casilla de verificación.

Propiedad	Descripción
Caption	Texto descriptivo que aparece junto a la casilla de verificación.
Enabled	Especifica si un usuario puede elegir o no la casilla de verificación.
ForeColor	Color del texto del título.
Left	Posición del extremo izquierdo de la casilla de verificación.
MousePointer	Apariencia del puntero del *mouse* (ratón) cuando está situado sobre la casilla de verificación.
Top	Posición de la parte superior de la casilla de verificación.

Visible	Especifica si la casilla de verificación es visible o no.

Los objetos tienen eventos y métodos asociados

Cada objeto reconoce y puede responder a determinadas acciones denominadas eventos. Un evento es una actividad específica y predeterminada, iniciada por el usuario o por el sistema. Los eventos, en la mayor parte de los casos, se generan por interacción del usuario. Por ejemplo, con un teléfono, se desencadena un evento cuando un usuario descuelga el receptor. Los eventos también se desencadenan cuando el usuario presiona los botones para efectuar una llamada.

En Visual FoxPro, las acciones del usuario que desencadenan eventos incluyen clics, movimientos del *mouse* y pulsaciones de teclas. Inicializar un objeto y encontrar una línea de código que produce un error son eventos iniciados por el sistema.

Los métodos son procedimientos asociados a un objeto. Los métodos se diferencian de los procedimientos normales de Visual FoxPro en que están vinculados inseparablemente a un objeto y tienen nombres distintos que los procedimientos normales de Visual FoxPro.

Los eventos pueden tener métodos asociados. Por ejemplo, si escribe código de método para el evento Click, ese código se ejecutará cuando se produzca el evento Click. Los métodos también pueden existir independientemente de los eventos. Se debe llamar a estos métodos de forma explícita en el código.

El conjunto de eventos es limitado, aunque amplio. No es posible crear nuevos eventos. Sin embargo, el conjunto de métodos puede ampliarse indefinidamente.

La tabla siguiente muestra algunos de los eventos asociados a una casilla de verificación:

Evento	Descripción
Click	El usuario hace clic en la casilla de verificación.
GotFocus	El usuario activa la casilla de verificación al hacer clic en ella o al llegar a ella a través de la tecla TAB.
LostFocus	El usuario selecciona otro control.

La tabla siguiente muestra algunos métodos asociados a una casilla de verificación:

Método	Descripción
Refresh	El valor de la casilla de verificación se actualiza para reflejar los cambios que se puedan haber producido en el origen de datos subyacente.
SetFocus	El enfoque se establece en la casilla de verificación como si el usuario hubiera presionado la tecla TAB hasta activar la casilla de verificación.

Consulte el capítulo 4, Descripción del modelo de eventos si desea obtener una explicación del orden

en que se producen los eventos.

Descripción de las clases de Visual FoxPro

Todas las propiedades, eventos y métodos de un objeto se especifican en la definición de clase. Además, las clases tienen las siguientes características que las hacen especialmente útiles para crear código reutilizable y fácil de mantener:

- Encapsulamiento
- Subclases
- Herencia

Ocultar la complejidad innecesaria

Cuando instale un teléfono en la oficina, lo más probable es que no le interese el funcionamiento interno del aparato para la recepción de llamadas, la realización o la finalización de conexiones con centralitas electrónicas o la conversión de las pulsaciones de tecla en señales electrónicas. Lo único que necesitará saber es que puede levantar el auricular, marcar los números apropiados y hablar con la persona con la que desea hablar. La complejidad de realizar esa conexión queda oculta. La ventaja de ignorar los detalles internos de un objeto para poder centrarse en los aspectos del objeto que necesita utilizar se denomina abstracción.

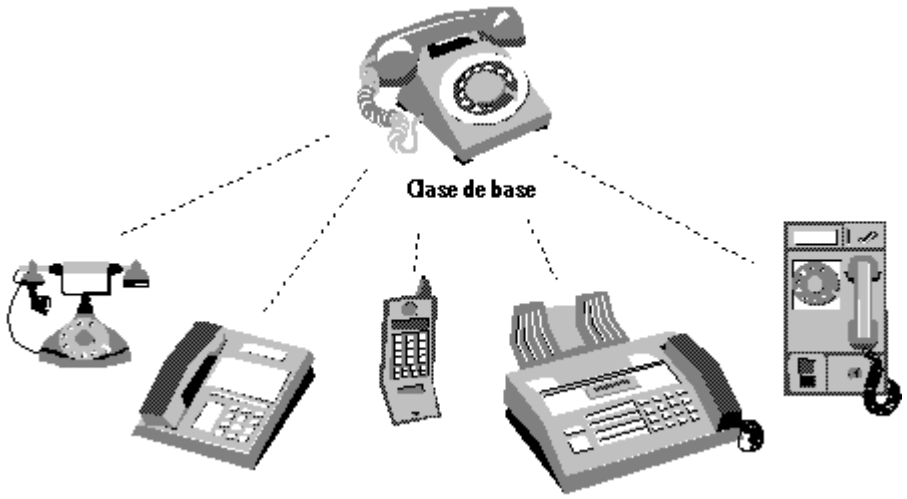
La complejidad interna puede estar oculta

El encapsulamiento, que empaqueta el código de métodos y propiedades en un mismo objeto, contribuye a la abstracción. Por ejemplo, las propiedades que determinan los elementos de un cuadro de lista y el código que se ejecuta al elegir un elemento de la lista pueden encapsularse en un único control que se agrega a un formulario.

Aprovechar la potencia de las clases existentes

Una subclase puede tener toda la funcionalidad de una clase existente, además de la funcionalidad y los controles adicionales que quiera darle. Si la clase es un teléfono básico, podrá tener subclases que tengan toda la funcionalidad del teléfono original y todas las características especializadas que desee darles.

Las subclases le permiten reutilizar código.



La creación de subclases es un modo de reducir la cantidad de código que hay que escribir. Puede comenzar definiendo un objeto que sea similar al deseado y personalizarlo.

Simplificar el mantenimiento de código

Con la herencia, si realiza un cambio en una clase, ese cambio se reflejará en todas las subclases que se basen en ella. Esta actualización automática ahorra tiempo y trabajo. Por ejemplo, si un fabricante de teléfonos quisiera cambiar los teléfonos de tipo marcación por aparatos de pulsación, se ahorraría mucho trabajo si pudiera hacer el cambio en el diagrama original y hacer que todos los teléfonos fabricados anteriormente con ese diagrama heredaran automáticamente la nueva característica, en lugar de tener que agregarla a todos los teléfonos existentes individualmente.

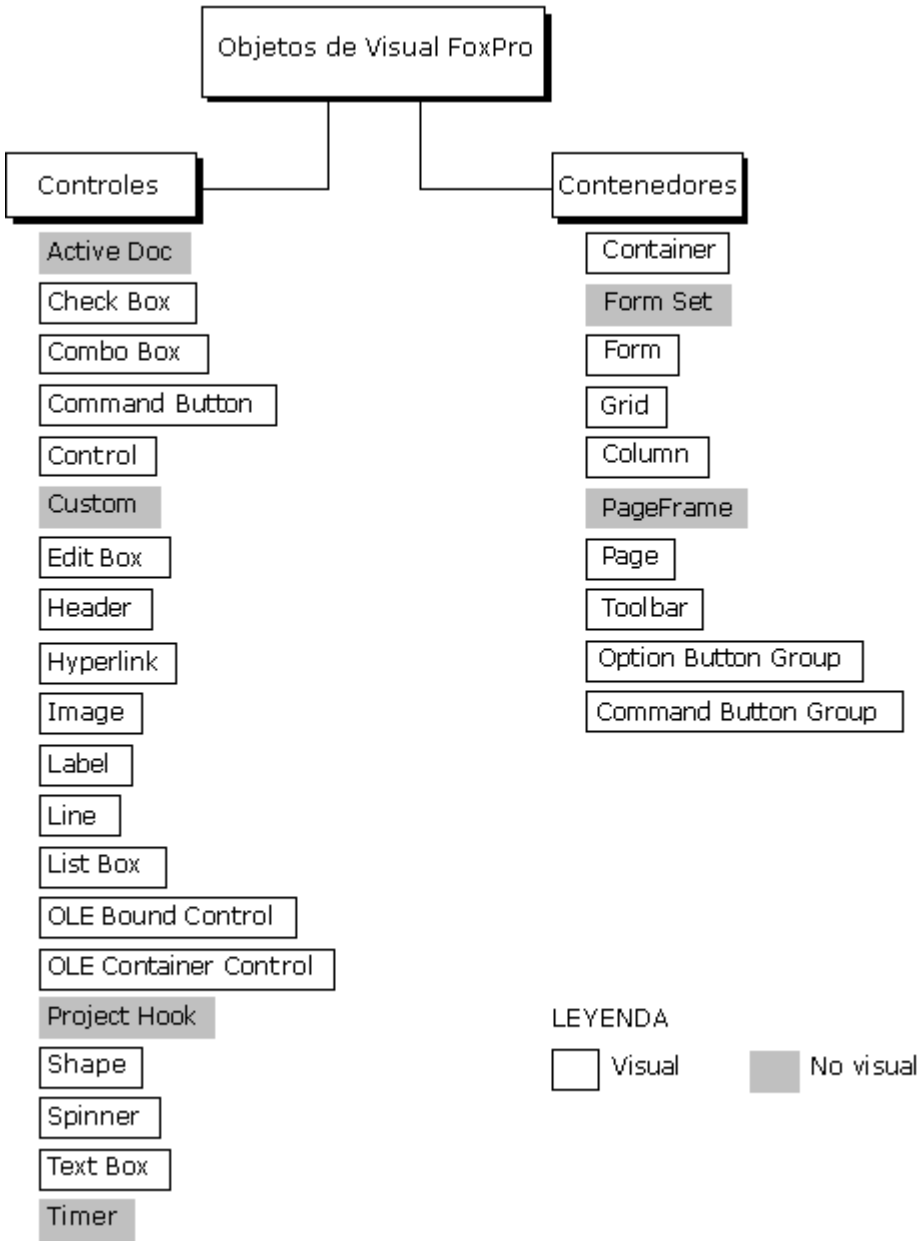
La herencia facilita el mantenimiento del código.

La herencia no funciona con el hardware, pero sí en el software. Si descubre un error en una clase, en lugar de tener que cambiar el código de todas las subclases podrá corregirlo una única vez en la clase y el cambio se propagará a todas las subclases pertenecientes a ella.

Jerarquía de clases de Visual FoxPro

A la hora de crear clases definidas por el usuario, resulta útil comprender la jerarquía de clases de Visual FoxPro.

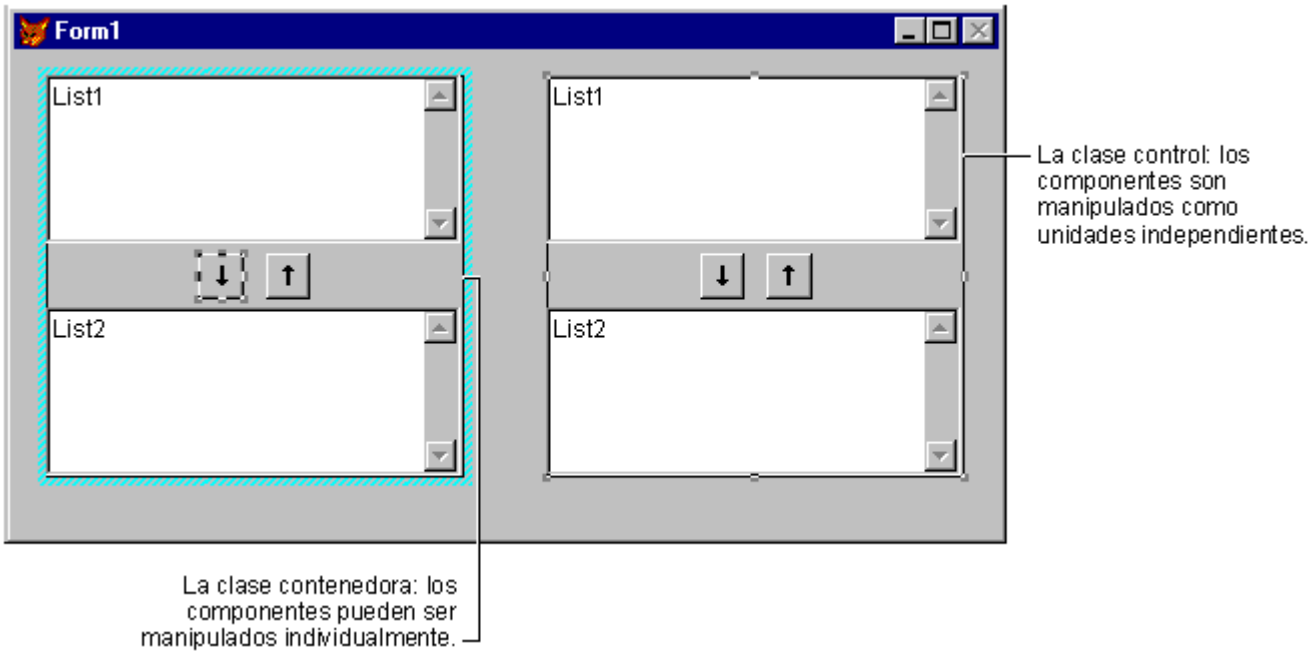
Jerarquía de clases de Visual FoxPro



Contenedores y no contenedores

Los dos tipos principales de clases de Visual FoxPro y por extensión, de objetos de Visual FoxPro, son las clases de contenedor y las clases de control.

Clases de contenedor y clases de control



Clases de contenedor

Los contenedores pueden incluir otros objetos y permiten el acceso a los objetos que contienen. Por ejemplo, si crea una clase de contenedor que consta de dos cuadros de lista y dos botones de comando y, a continuación, agrega a un formulario un objeto basado en esta clase, cada objeto individual podrá manipularse en tiempo de ejecución y en tiempo de diseño. Puede cambiar fácilmente las posiciones de los cuadros de lista o los títulos de los botones de comando. También puede agregar objetos al control en tiempo de diseño; por ejemplo, puede agregar etiquetas para identificar los cuadros de lista.

La tabla siguiente muestra los posibles componentes de cada clase de contenedor:

Contenedor	Puede contener
Grupos de botones de comando	Botones de comando
Contenedor	Cualquier control
Control	Cualquier control
Personalizado	Cualquier control, marcos de página, contenedor, personalizado
Conjuntos de formularios	Formularios, barras de herramientas
Formularios	Marcos de página, cualquier control, contenedores, personalizado
Columnas de cuadrícula	Encabezados de columnas, cualquier objeto excepto conjuntos de formularios, formularios, barras de herramientas, cronómetros y otras columnas

Cuadrículas	Columnas de cuadrícula
Grupos de botones de opción	Botones de opción
Marcos de página	Páginas
Páginas	Cualquier control, contenedores, personalizado
Proyecto	Archivos, servidores
Barras de herramientas	Cualquier control, marcos de página, contenedor

Clases de control

Las clases de control están más encapsuladas que las clases de contenedor, pero por esa misma razón es posible que sean menos flexibles. Las clases de control no tienen un método AddObject.

Adaptar la clase a la tarea

Es conveniente poder usar clases en muchos contextos distintos. Un diseño inteligente le permitirá decidir con mayor efectividad qué clases desea diseñar y qué funcionalidad va a incluir en la clase.

Decidir cuándo crear clases

Puede crear una clase para cada control y cada formulario que utilice, aunque éste no es el modo más efectivo de diseñar aplicaciones. Es muy probable que acabe con múltiples clases que tengan prácticamente la misma función y que deban mantenerse por separado.

Encapsular funcionalidad genérica

Cree una clase de control para funcionalidad genérica. Por ejemplo, los botones de comando que permiten al usuario mover el puntero de registro en una tabla, un botón para cerrar un formulario y un botón de ayuda pueden guardarse como clases y agregarse a formularios en cualquier momento que desee que los formularios tengan esta funcionalidad.

Puede exponer las propiedades y los métodos en una clase de modo que el usuario pueda integrarlos en el entorno de datos concreto de un formulario o un conjunto de formularios.

Proporcionar una apariencia y un uso coherentes

Puede crear clases de conjunto de formularios, de formulario y de control con una apariencia característica, de modo que todos los componentes de la aplicación tengan la misma apariencia. Por ejemplo, podría agregar gráficos y patrones de color específicos a una clase de formulario y utilizarla como plantilla para todos los formularios que cree. Podría crear una clase de cuadro de texto con una apariencia característica, como un efecto de sombreado, y usar esta clase en la aplicación en cualquier momento que desee agregar un cuadro de texto.

Decidir qué tipo de clase va a crear

Visual FoxPro permite crear distintos tipos de clases, cada uno con sus propias características. Especifique el tipo de clase que desea crear en el cuadro de diálogo Nueva clase o en la cláusula AS del comando CREATE CLASS.

Clases de base de Visual FoxPro

En el Diseñador de clases puede crear subclases para la mayoría de las clases de base de Visual FoxPro.

Clases de base de Visual FoxPro

ActiveDoc	Custom	Label	PageFrame
CheckBox	EditBox	Line	ProjectHook
Column*	Form	ListBox	Separator
CommandButton	FormSet	OLEBoundControl	Shape
CommandGroup	Grid	OLEContainerControl	Spinner
ComboBox	Header*	OptionButton*	TextBox
Container	Hyperlink Object	OptionGroup	Timer
Control	Image	Page*	ToolBar

* Estas clases son parte integral de un contenedor primario y no pueden usarse como subclases en el Diseñador de clases.

Todas las clases de base de Visual FoxPro reconocen el siguiente conjunto mínimo de eventos:

Evento	Descripción
Init	Ocurre cuando se crea el objeto.
Destroy	Ocurre cuando el objeto se libera de la memoria.
Error	Ocurre siempre que tiene lugar un error en procedimientos de evento o de método de la clase.

Todas las clases de base de Visual FoxPro tienen el siguiente conjunto mínimo de propiedades:

Propiedad	Descripción
Class	El tipo de clase de que se trata.
BaseClass	La clase de base de la que se deriva, como Form, Commandbutton, Custom, etc.

ClassLibrary	La biblioteca de clases en la que está almacenada.
ParentClass	La clase de la que se deriva la clase actual. Si la clase se deriva directamente de una clase de base de Visual FoxPro, la propiedad ParentClass es la misma que la propiedad BaseClass.

Extensión de las clases de base de Visual FoxPro

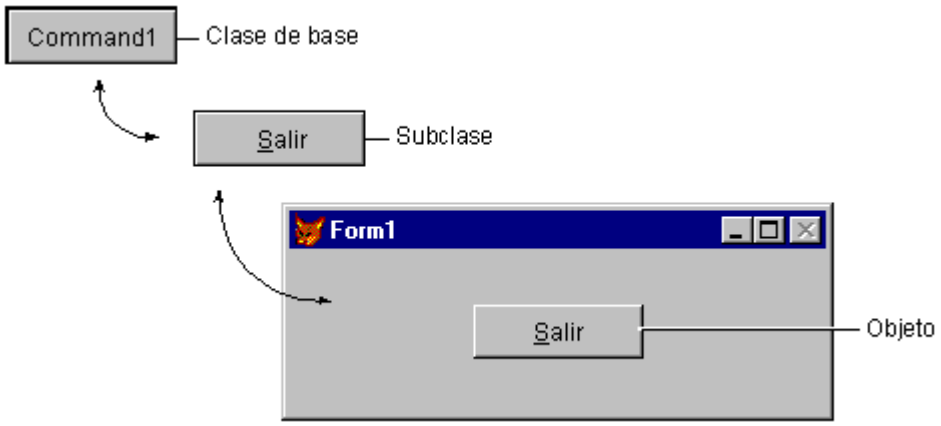
Puede convertir en subclases estas clases para establecer sus propias propiedades de control predeterminadas. Por ejemplo, si quiere que los nombres predeterminados de controles que agregue a formularios de sus aplicaciones reflejen automáticamente sus convenciones de nombres, puede crear clases basadas en las clases de base de Visual FoxPro para hacerlo. Puede crear clases de formulario con una apariencia o un comportamiento personalizado para que sirvan como plantillas para todos los formularios que cree.

También podría convertir en subclases las clases de base de Visual FoxPro para crear controles con funcionalidad encapsulada. Si quiere que un botón libere formularios cuando haga clic en él, puede crear una clase basada en la clase de botón de comando de Visual FoxPro, establecer como título "Salir" e incluir el siguiente comando en el evento Click:

```
THISFORM.Release
```

Puede agregar este nuevo botón a cualquier formulario de la aplicación.

Botón de comando personalizado agregado a un formulario



Crear controles con múltiples componentes

Las subclases no están limitadas a clases de base únicas. Puede agregar múltiples controles a una única definición de clase de contenedor. Muchas de las clases de la biblioteca de clases de ejemplo de Visual FoxPro están incluidas en esta categoría. Por ejemplo, la clase VCR de Buttons.vcx, ubicada en la carpeta ...\Samples\Vfp98\Classes de Visual Studio, contiene cuatro botones de comando para desplazarse por los registros de una tabla.

Creación de clases no visuales

Una clase basada en la clase personalizada de Visual FoxPro no tiene un elemento visual de tiempo de ejecución. Puede crear métodos y propiedades para la clase personalizada en el entorno del Diseñador de clases. Por ejemplo, podría crear una clase personalizada llamada `StrMethods` e incluir en ella una serie de métodos para manipular cadenas de caracteres. Podría agregar esta clase a un formulario con un cuadro de edición y llamar a los métodos cuando lo necesitara. Si tuviera un método llamado `WordCount`, podría llamarlo cuando lo necesitara:

```
THISFORM.txtCount.Value = ;
  THISFORM.StrMethods.WordCount(THISFORM.edtText.Value)
```

Las clases no visuales (como el control personalizado y el control cronómetro) tienen una representación visual, únicamente en tiempo de diseño, en el Diseñador de formularios. Establezca la propiedad de imagen de la clase personalizada como el archivo .bmp que desea mostrar en el Diseñador de formularios cuando se agregue la clase personalizada a un formulario.

Crear clases

Puede crear nuevas clases en el Diseñador de clases y puede ver cómo verá el usuario cada objeto a medida que lo diseña.

Para crear una clase nueva

- En el Administrador de proyectos, seleccione la ficha **Clases** y elija **Nuevo**.

  –O bien–

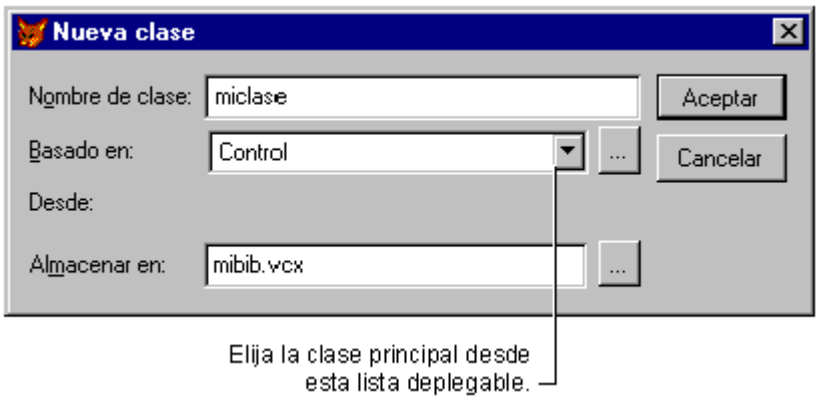
- En el menú **Archivo**, elija **Nuevo**, seleccione **Clase** y elija **Nuevo archivo**.

  –O bien–

- Utilice el comando CREATE CLASS.

El cuadro de diálogo **Nueva clase** le permite especificar el nombre de la nueva clase, la clase en la que se basa la nueva clase y la biblioteca en la que se almacenará.

Crear una clase nueva



Modificar una definición de clase

Cuando haya creado una clase, podrá modificarla. Los cambios realizados a una clase afectan a todas las subclases y a todos los objetos basados en esta clase. Puede agregar una mejora a una clase o reparar un error en la clase, y todas las subclases y los objetos basados en dicha clase heredarán el cambio.

Para modificar una clase en el Administrador de proyectos

1. Seleccione la clase que desea modificar.
2. Elija **Modificar**.

   Se abrirá el **Diseñador de clases**.

También puede modificar una definición de clase visual mediante el comando MODIFY CLASS.

Importante No cambie la propiedad Name de una clase si la usan otros componentes de la aplicación. De lo contrario, Visual FoxPro no podrá encontrar la clase cuando la necesite.

Subclases de una definición de clase

Hay dos formas de crear una subclase de una clase definida por el usuario.

Para crear una subclase de una clase definida por el usuario

1. En el cuadro de diálogo Nueva clase, haga clic en el botón de tres puntos situado a la derecha del cuadro **Basada en**.
2. En el cuadro de diálogo **Abrir**, elija la clase en la que desea basar la nueva clase.

   –O bien–

- Utilice el comando CREATE CLASS.

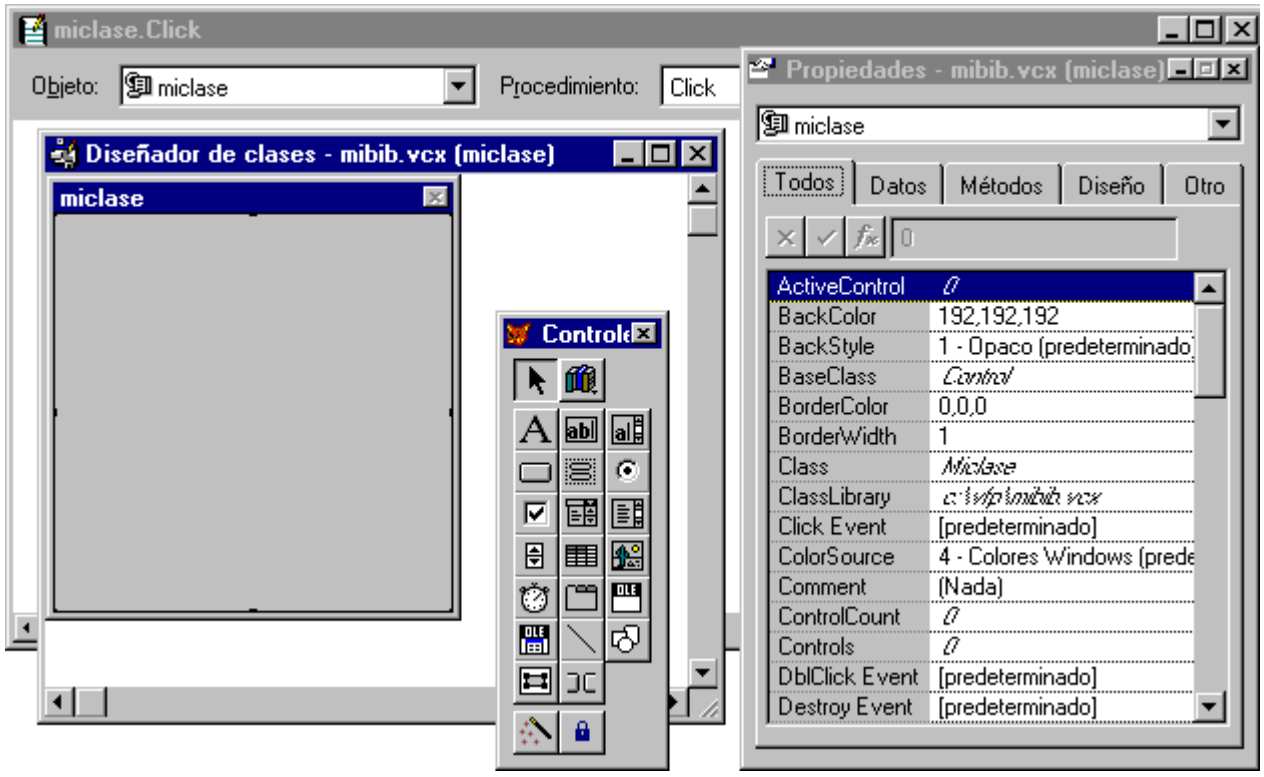
  Por ejemplo, para basar una nueva clase, `x`, en `parentclass` de Mylibrary.vcx, use el código siguiente:

```
CREATE CLASS x OF y AS parentclass ;
    FROM mylibrary
```

Utilizar el Diseñador de clases

Cuando especifica la clase en la que está basada la nueva clase y la biblioteca en la que se va a almacenar, se abre el Diseñador de clases.

Diseñador de clases



El Diseñador de clases proporciona la misma interfaz que el Diseñador de formularios, que le permite ver y modificar las propiedades de la clase en la ventana Propiedades. La ventana de edición de código le permite escribir código para que se ejecute cuando ocurran eventos o se llame a métodos.

Agregar objetos a una clase de control o a una clase de contenedor

Si basa la nueva clase en una clase de control o en una clase de contenedor, podrá agregarle controles del mismo modo que en el Diseñador de formularios: elija el botón del control en la barra de herramientas Controles de formularios y arrastre para ajustar el tamaño en el Diseñador de clases.

Independientemente del tipo de clase en el que base la nueva clase, puede establecer propiedades y escribir código de método. También podrá crear nuevas propiedades y métodos para la clase.

Agregar propiedades y métodos a una clase

Puede agregar tantas propiedades y métodos nuevos a la nueva clase como desee. Las propiedades contienen valores, mientras que los métodos contienen código de procedimiento que se ejecutará cuando llame al método.

Crear propiedades y métodos nuevos

Cuando crea propiedades y métodos nuevos para clases, las propiedades y los métodos tienen como alcance la clase, no los componentes individuales de la misma.

Para agregar una propiedad nueva a una clase

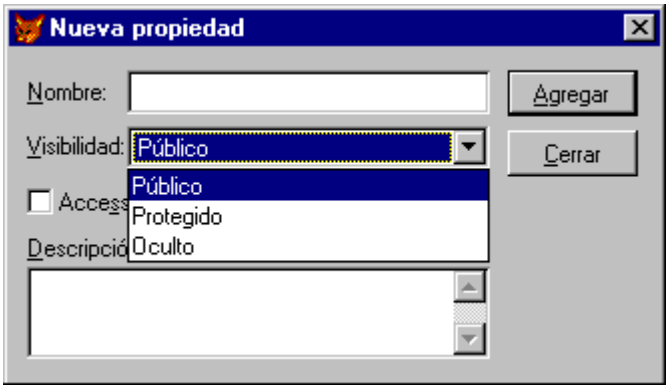
1. En el menú **Clase**, elija **Nueva propiedad**.

2. En el cuadro de diálogo Nueva propiedad, escriba el nombre de la propiedad.

3. Especifique la visibilidad: **Public**, **Protected** o **Hidden**.

   Puede tener acceso a una propiedad Public desde cualquier lugar de la aplicación. Las propiedades Protected y Hidden se tratan en "Proteger y ocultar miembros de clase" más adelante en este mismo capítulo.

   Cuadro de diálogo Nueva propiedad



4. Elija **Agregar**.

   También puede incluir una descripción de la propiedad que aparecerá en la parte inferior de la ventana Propiedades en el **Diseñador de clases** y en el **Diseñador de formularios** cuando se agregue el control a un formulario.

Solución de problemas Cuando agregue una propiedad a una clase que un usuario de la clase pueda establecer, el usuario puede introducir un valor incorrecto para la propiedad que cause errores en tiempo de ejecución. Tiene que documentar de forma explícita los valores válidos de la propiedad. Si la propiedad puede establecerse como 0, 1 ó 2, por ejemplo, indíquelo en el cuadro **Descripción** del cuadro de diálogo Nueva propiedad. También es conveniente comprobar el valor de la propiedad en código que haga referencia a ella.

Para crear una propiedad de matriz

- En el cuadro **Nombre** del cuadro de diálogo Nueva propiedad, especifique el nombre, el tamaño y las dimensiones de la matriz.

  Por ejemplo, para crear una propiedad de matriz llamada `mimatriz` con diez filas y dos columnas, escriba lo siguiente en el cuadro Nombre:

  ```
  mimatriz[10,2]
  ```

La propiedad de matriz es de sólo lectura en tiempo de diseño y se muestra en cursiva en la ventana Propiedades. Se puede administrar y redimensionar en tiempo de ejecución. Para ver un ejemplo del

uso de una propiedad de matriz, consulte "Administrar varias instancias de un formulario" en el capítulo 9, Crear formularios.

Para agregar un método nuevo a una clase

1. En el menú **Clase**, elija **Nuevo método**.
2. En el cuadro de diálogo Nuevo método, escriba el nombre del método.
3. Especifique la visibilidad: **Public**, **Protected** o **Hidden**.
4. Seleccione la casilla de verificación Access para crear un método de Access, seleccione la casilla de verificación para crear un método Assign o seleccione ambas casillas de verificación para crear métodos Access y Assign.

Los métodos Access y Assign le permiten ejecutar código cuando se consulta el valor de una propiedad o cuando se intenta cambiar su valor.

El código de un método Access se ejecuta cuando se consulta el valor de una propiedad, generalmente al utilizar la propiedad en una referencia de objeto, al almacenar el valor de una propiedad en una variable o al mostrar el valor de la propiedad con un signo de interrogación (?).

El código de un método Assign se ejecuta cuando intenta modificar el valor de una propiedad, generalmente mediante los comandos STORE o = para asignar un nuevo valor a la propiedad.

Para obtener más información acerca de los métodos Access y Assign, consulte Métodos Access y Assign.

También puede incluir una descripción del método.

Proteger y ocultar miembros de clases

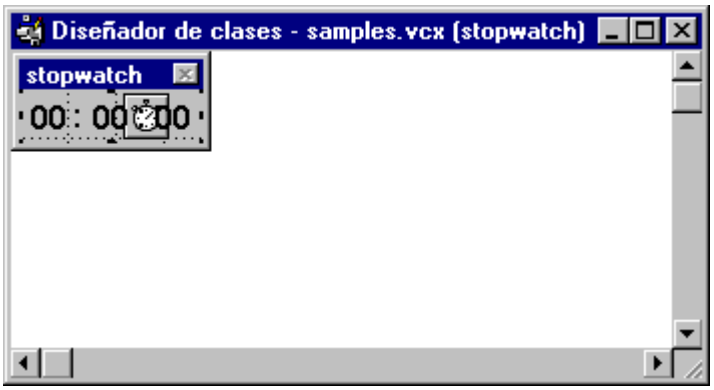
Las propiedades y métodos de una definición de clase son **Public** de forma predeterminada: el código de otras clases u otros procedimientos puede establecer las propiedades o llamar a los métodos. A las propiedades y los métodos definidos como **Protected** sólo pueden tener acceso otros métodos de la definición de la clase o de subclases de la clase. A las propiedades y los métodos definidos como **Hidden** sólo pueden tener acceso otros miembros de la definición de la clase. Las subclases de la clase no pueden "ver" o hacer referencia a miembros ocultos.

Para asegurar un correcto funcionamiento en algunas clases, deberá impedir que los usuarios cambien las propiedades o llamen al método desde fuera de la clase mediante programación.

El ejemplo siguiente ilustra el uso de propiedades y métodos protegidos de una clase

La clase Stopwatch incluida en Samples.vcx, en el directorio ...\Samples\Vfp98\Classes de Visual Studio, incluye un cronómetro y cinco etiquetas que muestran el tiempo transcurrido:

La clase Stopwatch de Samples.vcx



La clase Stopwatch contiene etiquetas y un cronómetro.

Valores de las propiedades de la clase Stopwatch

Control	Propiedad	Valor
lblSeconds	Caption	00
lblColon1	Caption	:
lblMinutes	Caption	00
lblColon2	Caption	:
lblHours	Caption	00
tmrSWatch	Interval	1000

Esta clase tiene también tres propiedades protegidas, nSec, nMin y nHour, así como un método protegido, UpdateDisplay. Los otros tres métodos personalizados de la clase, Start, Stop y Reset no están protegidos.

Sugerencia Elija **Información de clase** en el menú **Clase** para ver la visibilidad de todas las propiedades y métodos de una clase.

Las propiedades protegidas se utilizan en cálculos internos en el método UpdateDisplay y el evento Timer. El método UpdateDisplay establece los títulos de las etiquetas para que reflejen el tiempo transcurrido.

Método UpdateDisplay

Código	Comentarios

`CSecDisplay = ALLTRIM(STR(THIS.nSec))`
`cMinDisplay = ALLTRIM(STR(THIS.nMin))`
`cHourDisplay = ALLTRIM(STR(THIS.nHour))`
	Convierte las propiedades numéricas al tipo Character para mostrarlas en los títulos de etiqueta.

`THIS.lblSeconds.Caption = ;`	Establece los títulos de etiqueta,

`IIF(THIS.nSec < 10, ;`
`"0" ,"") + cSecDisplay`
`THIS.lblMinutes.Caption = ;`
`IIF(THIS.nMin < 10, ;`
`"0", "") + cMinDisplay`
`THIS.lblHours.Caption = ;`
`IIF(THIS.nHour < 10, ;`
`"0", "") + cHourDisplay`
	conservando los 0 iniciales si el valor de la propiedad numérica es menor que 10.

La tabla siguiente muestra el código del evento `tmrSWatch.Timer`.

Evento Timer

Código	Comentarios

`THIS.Parent.nSec = THIS.Parent.nSec + 1`
`IF THIS.Parent.nSec = 60`
`THIS.Parent.nSec = 0`
`THIS.Parent.nMin = ;`
`THIS.Parent.nMin + 1`
`ENDIF`
	Incrementa el valor de la propiedad `nSec` cada vez que se desencadena el evento de cronómetro cada segundo.
	Si `nSec` ha llegado a 60, lo restablece a 0 e incrementa la propiedad `nMin`.

`IF THIS.Parent.nMin = 60`
`THIS.Parent.nMin = 0`
`THIS.Parent.nHour = ;`
`THIS.Parent.nHour + 1`
`ENDIF`
`THIS.Parent.UpdateDisplay`
	Si `nMin` ha llegado a 60, lo restablece a 0 e incrementa la propiedad `nHour`.
	
	Llama al método `UpdateDisplay` cuando se establecen los nuevos valores de la propiedad.

La clase Stopwatch tiene tres métodos que no están protegidos: Start, Stop y Reset. Un usuario debe poder llamar directamente a estos métodos para controlar el cronómetro.

El método Start contiene la línea de código siguiente:

```
THIS.tmrSWatch.Enabled = .T.
```

El método Stop contiene la línea de código siguiente:

```
THIS.tmrSWatch.Enabled = .F.
```

El método Reset establece las propiedades protegidas a 0 y llama al método protegido:

```
THIS.nSec = 0
THIS.nMin = 0
THIS.nHour = 0
THIS.UpdateDisplay
```

El usuario no puede establecer directamente estas propiedades o llamar a este método, pero el código del método Reset sí puede hacerlo.

Especificar el valor predeterminado para una propiedad

Al crear una nueva propiedad, su valor predeterminado es falso (.F.). Para especificar un valor predeterminado distinto para una propiedad, utilice la ventana Propiedades. En la ficha **Otras**, haga clic en la propiedad y establezca el valor deseado. Este será el valor inicial de la propiedad cuando se agregue la clase a un formulario o a un conjunto de formularios.

También puede establecer cualquiera de las propiedades de clase de base en el Diseñador de clases. Cuando un objeto basado en la clase se agregue al formulario, reflejará el valor de su propiedad en lugar del valor de la propiedad de la clase de base de Visual FoxPro.

Sugerencia Si desea convertir el valor predeterminado de una propiedad en una cadena vacía, seleccione el valor en el cuadro **Edición de propiedades** y presione la tecla RETROCESO.

Especificar la apariencia en tiempo de diseño

Puede especificar el icono de barra de herramientas y el de contenedor para su clase en el cuadro de diálogo Información de clase.

Para establecer un icono de barra de herramientas para una clase

1. En el Diseñador de clases, elija **Información de clase** en el menú **Clase**.

2. En el cuadro de diálogo Información de clase, escriba el nombre y la ruta de acceso del archivo .BMP en el cuadro **Icono de la barra de herramientas**.

   Sugerencia El archivo de mapa de bits (archivo .bmp) para el icono de la barra de herramientas debe tener 15 por 16 píxeles. Si la imagen es mayor o menor, se ajustará a 15 por 16 píxeles y posiblemente no tendrá la apariencia deseada.

El icono de barra de herramientas especificado se muestra en la barra de herramientas Controles de formularios cuando se llena la barra de herramientas con las clases de la biblioteca de clases.

También puede especificar que se muestre el icono para la clase en el Administrador de proyectos y el Examinador de clases si establece el icono contenedor.

Para establecer un icono contenedor para una clase

1. En el Diseñador de clases, elija **Información de clase** en el menú **Clase**.

2. En el cuadro **Icono de contenedor**, escriba el nombre y la ruta de acceso del archivo .bmp que se va a mostrar en el botón de la barra de herramientas Controles de formularios.

Usar archivos de bibliotecas de clases

Todas las clases diseñadas visualmente se almacenan en una biblioteca de clases con la extensión de archivo .vcx.

Crear una biblioteca de clases

Una biblioteca de clases puede crearse de una de estas tres formas.

Para crear una biblioteca de clases

- Cuando cree una clase, especifique un nuevo archivo de biblioteca de clases en el cuadro **Almacenar en** del cuadro de diálogo Nueva clase.

  –O bien–

- Utilice el comando CREATE CLASS, especificando el nombre de la nueva biblioteca de clases.

  Por ejemplo, la instrucción siguiente crea una nueva clase llamada `miclase` y una nueva biblioteca de clases llamada `nue_bib`:

  ```
  CREATE CLASS miclase OF nue_bib AS CUSTOM
  ```

  –O bien–

- Utilice el comando CREATE CLASSLIB.

  Por ejemplo, escriba el comando siguiente en la ventana Comandos para crear una biblioteca de clases llamada `nue_bib`:

  ```
  CREATE CLASSLIB nue_bib
  ```

Copiar y quitar clases de bibliotecas de clases

Cuando agregue una biblioteca de clases a un proyecto, podrá copiar clases de una biblioteca a otra con facilidad o, simplemente, quitar clases de las bibliotecas.

Para copiar una clase de una biblioteca a otra

1. Asegúrese de que ambas bibliotecas están en un proyecto (no necesariamente en el mismo).
2. En el Administrador de proyectos, seleccione la ficha **Clases**.
3. Haga clic en el signo más (+) situado a la izquierda de la biblioteca de clases en la que se encuentra ahora la clase.
4. Arrastre la clase desde la biblioteca original y colóquela en la nueva.

   Sugerencia Es conveniente guardar en una biblioteca de clases una clase y todas las subclases basadas en ella. Si tiene una clase que contiene elementos de muchas bibliotecas de clases distintas, estas bibliotecas deberán estar abiertas, por lo que se tardará un poco más en cargar inicialmente la clase en tiempo de ejecución y en tiempo de diseño.

Para quitar una clase de una biblioteca

- Seleccione la clase en el Administrador de proyectos y elija **Quitar**.

  –O bien–

- Utilice el comando REMOVE CLASS.

Puede utilizar el comando RENAME CLASS para cambiar el nombre de una clase de una biblioteca de clases. Sin embargo, recuerde que cuando cambia el nombre de una clase, los formularios que contienen la clase y las subclases en otros archivos .vcx siguen haciendo referencia al nombre antiguo y no volverán a funcionar correctamente.

Visual FoxPro incluye un Examinador de clases para facilitar el uso y la administración de clases y bibliotecas de clases. Para obtener más información, vea la ventana Examinador de clases.

Agregar clases a formularios

Puede arrastrar una clase desde el Administrador de proyectos hasta el Diseñador de formularios o hasta el Diseñador de clases. También puede registrar las clases de modo que puedan mostrarse directamente en la barra de herramientas Controles de formularios del Diseñador de formularios o el Diseñador de clases y agregarse a contenedores de la misma forma que los controles estándar.

Para registrar una biblioteca de clases

1. En el menú **Herramientas**, elija **Opciones**.
2. En el cuadro de diálogo **Opciones**, elija la ficha Controles.
3. Seleccione **Bibliotecas de clases visuales** y elija **Agregar**.
4. En el cuadro de diálogo **Abrir**, elija una biblioteca de clases para agregar el registro y, a continuación, elija **Abrir**.
5. Elija **Establecer como predeterminado** si desea que la biblioteca de clases esté disponible en la barra de herramientas Controles de formularios en sesiones futuras de Visual FoxPro.

También puede agregar la biblioteca de clases a la barra de herramientas Controles de formularios si elige **Agregar** en el submenú del botón **Ver clases**. Para que estas clases estén disponibles en la barra de herramientas Controles de formularios en sesiones futuras de Visual FoxPro, tendrán que establecer el valor predeterminado en el cuadro de diálogo Opciones.

Anular valores predeterminados de propiedades

Al agregar a un formulario objetos basados en una clase definida por el usuario, puede cambiar el valor de todas las propiedades de la clase que no estén protegidas, anulando los valores predeterminados. Si posteriormente cambia las propiedades de clase en el Diseñador de clases, no se verá afectada la configuración del objeto del formulario. Si no ha cambiado el valor de una propiedad del formulario y cambia el de la clase, el cambio también surtirá efecto en el objeto.

Por ejemplo, un usuario puede agregar a un formulario un objeto basado en su clase y cambiar la propiedad BackColor de blanco a rojo. Si cambia a verde la propiedad BackColor de la clase, el objeto del formulario del usuario seguirá teniendo un valor rojo para BackColor. Por otra parte, si el usuario no cambia la propiedad BackColor del objeto y usted cambia a verde el color de fondo de la clase, la propiedad BackColor del objeto del formulario heredará el cambio y también será verde.

Llamar al código de métodos de clase primaria

Un objeto o una clase que se basa en otra clase hereda automáticamente la funcionalidad de la clase original. Sin embargo, puede anular fácilmente el código de métodos heredado. Por ejemplo, puede escribir nuevo código para el evento Click de una clase después de haberla convertido en subclase o después de agregar al contenedor un objeto basado en la clase. En ambos casos, el nuevo código se ejecuta en tiempo de ejecución; el código original no se ejecuta.

Sin embargo, es más frecuente que quiera agregar funcionalidad a la nueva clase u objeto conservando la funcionalidad original. De hecho, una de las decisiones clave que tiene que hacer en la programación orientada a objetos es qué funcionalidad va a incluir a nivel de clase, a nivel de subclase y a nivel de objeto. Puede optimizar el diseño de la clase con la función DODEFAULT( ) o el operador de resolución de alcance (::) para agregar código a distintos niveles de la jerarquía del contenedor o de la clase.

Agregar funcionalidad a subclases

Puede llamar al código de la clase primaria desde una subclase mediante la función DODEFAULT( ).

Por ejemplo, `cmdOK` es una clase de botón de comando almacenada en Buttons.vcx, ubicada en el directorio ...\Samples\Vfp98\Classes de Visual Studio. El código asociado al evento Click de `cmdOk` libera el formulario que contiene el botón. `cmdCancel` es una subclase de `cmdOk` de la misma biblioteca de clases. Para agregar funcionalidad a `cmdCancel` para descartar cambios, por ejemplo, puede agregar el código siguiente al evento Click:

```
IF USED( ) AND CURSORGETPROP("Buffering") != 1
   TABLEREVERT(.T.)
ENDIF
DODEFAULT( )
```

Como los cambios se escriben en una tabla almacenada en búfer de forma predeterminada cuando se cierra la tabla, no tiene que agregar código TABLEUPDATE( ) a `cmdOk`. El código adicional de `cmdCancel` deshace los cambios realizados a la tabla antes de llamar al código de `cmdOk`, la clase primaria, para liberar el formulario.

Jerarquías de clases y de contenedores

Las jerarquías de clases y de contenedores son dos entidades distintas. Visual FoxPro busca código de evento en la jerarquía de clases, mientras que se hace referencia a los objetos en la jerarquía de contenedores. La siguiente sección, "Referencias a objetos en la jerarquía de contenedores", trata la jerarquía de contenedores. Más adelante en este capítulo se explican las jerarquías de clases en la sección Llamar a código de evento en la jerarquía de clases.

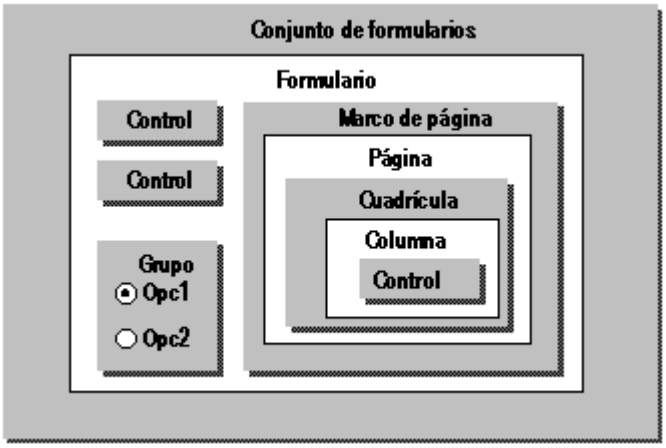
Referencias a objetos en la jerarquía de contenedores

Para manipular un objeto, hay que identificarlo en relación a la jerarquía de contenedores. Por ejemplo, para manipular un control de un formulario perteneciente a un conjunto de formularios, deberá hacer referencia al conjunto de formularios, al formulario y, por último, al control.

Hacer referencia a un objeto dentro de su jerarquía de contenedores se puede comparar con dar una dirección del objeto a Visual FoxPro. Cuando describe la ubicación de una casa a otra persona fuera de su marco inmediato de referencia, debe indicar el país, la provincia o la región, la ciudad, la calle o bien sólo el número de la calle donde se encuentra la vivienda, según lo lejos que se encuentre esa otra persona. De lo contrario, podría haber cierta confusión.

La ilustración siguiente muestra una posible situación de anidamiento del contenedor.

Contenedores anidados



Para desactivar el control de la columna de cuadrícula, deberá proporcionar la dirección siguiente:

```
Formset.Form.PageFrame.Page.;
   Grid.Column.Control.Enabled = .F.
```

La propiedad ActiveForm del objeto aplicación (_VFP) le permite manipular el formulario activo aunque no conozca su nombre. Por ejemplo, la siguiente línea de código cambia el color de fondo del formulario activo, independientemente del conjunto de formularios al que pertenezca:

```
_VFP.ActiveForm.BackColor = RGB(255,255,255)
```

De forma similar, la propiedad ActiveControl permite manipular el control activo del formulario activo. Por ejemplo, la expresión siguiente introducida en la ventana Inspección muestra el nombre del control activo de un formulario a medida que se eligen interactivamente los distintos controles:

```
_VFP.ActiveForm.ActiveControl.Name
```

Referencias relativas

Cuando haga referencia a objetos desde la jerarquía de contenedores (por ejemplo, en el evento Click de un botón de comando de un formulario perteneciente a un conjunto de formularios), puede utilizar algunos métodos abreviados para identificar el objeto que desea manipular. La tabla siguiente indica las propiedades o las palabras clave que facilitan la referencia a un objeto desde la jerarquía del objeto:

Propiedad o palabra clave	Referencia
Parent	El contenedor más inmediato del objeto
THIS	El objeto
THISFORM	El formulario que contiene el objeto
THISFORMSET	El conjunto de formularios que contiene el objeto

Nota Sólo puede utilizar THIS, THISFORM y THISFORMSET en código de métodos y eventos.

La tabla siguiente proporciona ejemplos del uso de THISFORMSET, THISFORM, THIS y Parent para establecer propiedades de objetos:

Comando	Dónde incluir el comando
`THISFORMSET.frm1.cmd1.Caption = "Aceptar"`	En el código de evento o de método de cualquier control de cualquier formulario del conjunto de formularios.
`THISFORM.cmd1.Caption = "Aceptar"`	En el código de evento o de método de cualquier control del mismo formulario en el que está `cmd1`.
`THIS.Caption = "Aceptar"`	En el código de evento o de método del control cuyo título desee cambiar.
`THIS.Parent.BackColor = RGB(192,0,0)`	En el código de evento o de método de un control de un formulario. El comando cambia a rojo oscuro el color de fondo del formulario.

Establecer propiedades

Las propiedades de un objeto pueden establecerse en tiempo de ejecución o en tiempo de diseño.

Para establecer una propiedad

- Utilice esta sintaxis:

  Contenedor.Objeto.Propiedad = Valor

Por ejemplo, las instrucciones siguientes establecen varias propiedades de un a cuadro de texto llamado `txtDate` en un formulario llamado `frmPhoneLog`:

```
frmPhoneLog.txtDate.Value = DATE( ) && Muestra la fecha actual
frmPhoneLog.txtDate.Enabled = .T.   && El control está activado
frmPhoneLog.txtDate.ForeColor = RGB(0,0,0)        && texto en negro
frmPhoneLog.txtDate.BackColor = RGB(192,192,192)  && fondo en gris
```

Para la configuración de propiedades de los ejemplos anteriores, `frmPhoneLog` es el objeto contenedor de mayor nivel. Si `frmPhoneLog` estuviera incluido en un conjunto de formularios, también debería incluir el conjunto de formularios en la ruta de acceso primaria:

```
frsContacts.frmPhoneLog.txtDate.Value = DATE( )
```

Establecer múltiples propiedades

La estructura WITH ... ENDWITH simplifica el establecimiento de múltiples propiedades. Por ejemplo, para establecer múltiples propiedades de una columna en una cuadrícula de un formulario perteneciente a un conjunto de formularios, podría utilizar la sintaxis siguiente:

```
WITH THISFORMSET.frmForm1.grdGrid1.grcColumn1
   .Width = 5
   .Resizable = .F.
   .ForeColor = RGB(0,0,0)
   .BackColor = RGB(255,255,255)
   .SelectOnEntry = .T.
ENDWITH
```

Llamar a métodos

Una vez creado un objeto, puede llamar a los métodos de ese objeto desde cualquier lugar de la aplicación.

Para llamar a un método

- Utilice esta sintaxis:

  Primario.Objeto.Método

Las instrucciones siguientes llaman a métodos para mostrar un formulario y establecer el enfoque en un cuadro de texto:

```
frsFormSet.frmForm1.Show
frsFormSet.frmForm1.txtGetText1.SetFocus
```

Los métodos que devuelven valores y se utilizan en expresiones deben terminar en paréntesis de apertura y de cierre. Por ejemplo, la instrucción siguiente establece el título de un formulario como el valor devuelto por el método definido por el usuario `GetNewCaption`:

```
Form1.Caption = Form1.GetNewCaption( )
```

Nota Los parámetros transferidos a métodos deben incluirse entre paréntesis después del nombre del método; por ejemplo, `Form1.Show(nStyle)`. transfiere `nStyle` al código del método Show de Form1.

Responder a eventos

El código incluido en un procedimiento de evento se ejecuta cuando se produce el evento. Por ejemplo, el código incluido en el procedimiento de evento Click de un botón de comando se ejecutará cuando el usuario haga clic en el botón de comando.

Puede activar los eventos Click, DblClick, MouseMove y DragDrop con el evento MOUSE o usar el comando ERROR para generar eventos Error y el comando KEYBOARD para generar eventos KeyPress. No puede hacer que se produzca ningún otro evento mediante programación, pero sí puede llamar al procedimiento asociado con el evento. Por ejemplo, la instrucción siguiente hace que se ejecute el código del evento Activate de `frmPhoneLog`, pero no activa el formulario:

```
frmPhoneLog.Activate
```

Si desea activar el formulario, utilice el método Show del formulario. Al llamar al método Show se mostrará y activará el formulario, momento en el que también se ejecutará el código del evento Activate:

```
frmPhoneLog.Show
```

Definir clases mediante programación

Las clases se pueden definir visualmente en el Diseñador de clases y el Diseñador de formularios o mediante programación en archivos .PRG. En esta sección se explica cómo escribir definiciones de clase. Para obtener información sobre comandos, funciones y operadores específicos, vea la Ayuda. Para obtener más información sobre formularios, consulte el capítulo 9, Crear formularios

En un archivo de programa es posible tener código de programa delante de las definiciones de clase, pero no después de ellas, del mismo modo que el código de programa no puede ir después de los procedimientos de un programa. El intérprete de comandos básico para la creación de clases tiene la sintaxis siguiente:

DEFINE CLASS *NombreClase1* AS *ClasePrimaria* [OLEPUBLIC]
[[PROTECTED | HIDDEN *NombrePropiedad1*, *NombrePropiedad2* ...]
[Object.]*NombrePropiedad* = *eExpresión* ...]
[ADD OBJECT [PROTECTED] *NombreObjeto* AS *NombreClase2* [NOINIT]
[WITH *cListaPropiedades*]]...
[[PROTECTED | HIDDEN] FUNCTION | PROCEDURE *Nombre*[_ACCESS | _ASSIGN]
[NODEFAULT]
cInstrucciones
[ENDFUNC | ENDPROC]]...
ENDDEFINE

Proteger y ocultar miembros de clase

Puede proteger u ocultar propiedades y métodos de una definición de clase con las palabras clave PROTECTED y HIDDEN del comando DEFINE CLASS.

Por ejemplo, si crea una clase para almacenar información sobre empleados y no desea que los usuarios puedan modificar la fecha de contratación, puede proteger la propiedad FechaContr. Si los usuarios necesitan averiguar cuándo se contrató a un empleado determinado, podrá incluir un método para devolver la fecha de contratación.

```
DEFINE CLASS empleado AS CUSTOM
PROTECTED FechaContr
    Nombre = ""
    Apellido = ""
    Dirección = ""
    FechaContr = { - - }

PROCEDURE ObtFechaContr
    RETURN This.FechaContr
ENDPROC
ENDDEFINE
```

Crear objetos a partir de clases

Cuando haya guardado una clase visual, puede crear un objeto basado en ella mediante la función CREATEOBJECT( ). El ejemplo siguiente muestra la ejecución de un formulario guardado como una definición de clase en el archivo de biblioteca de clases Forms.vcx:

Crear y mostrar un objeto Form cuya clase se diseñó en el Diseñador de formularios

Código	Comentarios
`SET CLASSLIB TO Forms ADDITIVE`	Establece como biblioteca de clases el archivo .vcx en el que se guardó la definición del formulario. La palabra clave ADDITIVE impide que este comando cierre otras bibliotecas de clases que estuvieran abiertas.
`frmTest = CREATEOBJECT ("FormPrueba")`	Este código supone que el nombre de la clase de formulario guardada en la biblioteca de clases es FormPrueba.
`frmTest.Show`	Muestra el formulario.

Agregar objetos a una clase contenedor

Puede utilizar la cláusula ADD OBJECT en el comando DEFINE CLASS o en el método AddObject para agregar objetos a un contenedor.

Por ejemplo, la siguiente definición de clase se basa en un formulario. El comando ADD OBJECT agrega dos botones de comando al formulario:

```
DEFINE CLASS miform AS FORM
    ADD OBJECT cmdOK AS COMMANDBUTTON
    ADD OBJECT PROTECTED cmdCancel AS COMMANDBUTTON
ENDDEFINE
```

Utilice el método AddObject para agregar objetos a un contenedor después de crear el objeto contenedor. Por ejemplo, las líneas de código siguientes crean un objeto formulario y le agregan dos botones de comando:

```
frmMessage = CREATEOBJECT("FORM")
frmMessage.AddObject("txt1", "TEXTBOX")
frmMessage.AddObject("txt2", "TEXTBOX")
```

También puede utilizar el método AddObject en el código de método de una clase. Por ejemplo, la definición de clase siguiente utiliza AddObject en el código asociado al evento Init para agregar un control a una columna de cuadrícula.

```
DEFINE CLASS micuad AS GRID
ColumnCount = 3
PROCEDURE Init
    THIS.Column2.AddObject("cboCliente", "COMBOBOX")
    THIS.Column2.CurrentControl = "cboCliente"
ENDPROC
ENDDEFINE
```

Agregar y crear clases en código de métodos

Puede agregar objetos a un contenedor mediante programación con el método AddObject. También puede crear objetos con la función CREATEOBJECT( ) en los métodos Load, Init o en cualquier otro método de la clase.

Cuando agregue un objeto con el método AddObject, el objeto se convierte en un miembro del contenedor. La propiedad Parent del objeto agregado se refiere al contenedor. Cuando un objeto basado en el contenedor o en la clase del control se libera de la memoria, también se libera el objeto agregado.

Cuando crea un objeto con la función CREATEOBJECT( ), el objeto está en el alcance de una propiedad de la clase o variable del método que llama a esta función. La propiedad primaria del objeto no está definida.

Asignar código de método y código de evento

Además de escribir código para los métodos y eventos de un objeto, puede ampliar el conjunto de métodos en las subclases de clases de base de Visual FoxPro. Estas son las reglas para escribir código de evento y métodos:

- El conjunto de eventos para las clases de base de Visual FoxPro es limitado y no puede ampliarse.

- Todas las clases reconocen un conjunto limitado de eventos predeterminados, que incluye como mínimo los eventos Init, Destroy y Error.
- Al crear en una definición de clase un método con el mismo nombre que un evento reconocible por la clase, el código del método se ejecutará cuando se produzca el evento.
- Puede agregar métodos a las clases mediante la creación de un procedimiento o una función en la definición de clase.

- Puede crear métodos Access y Assign para sus clases si crea un procedimiento o una función con el mismo nombre que una propiedad de clase y anexa _ACCESS o _ASSIGN al nombre de procedimiento o de función.

Llamar al código de evento en la jerarquía de clases

Al crear una clase, ésta hereda automáticamente todas las propiedades, los métodos y los eventos de la clase primaria. Si se escribe código para un evento en la clase primaria, ese código se ejecutará cuando se produzca el evento con respecto a un objeto basado en la subclase. Sin embargo, podrá sobrescribir el código de la clase primaria escribiendo código para el evento en la subclase.

Para llamar explícitamente al código de evento en una clase primaria cuando la subclase tiene código escrito para el mismo evento, utilice la función DODEFAULT( ).

Por ejemplo, podría tener una clase llamada `cmdBottom` basada en la clase de base del botón de comando que tuviera el código siguiente en el evento Click:

```
GO BOTTOM
THISFORM.Refresh
```

Al agregar un objeto basado en esta clase a un formulario llamado, por ejemplo, `cmdInferior1`, podría decidir que también desea mostrar un mensaje para informar al usuario de que el puntero de registro está en la parte inferior de la tabla. Podría agregar el código siguiente al evento Click del objeto para mostrar el mensaje:

```
WAIT WINDOW "En la parte inferior de la tabla" TIMEOUT 1
```

Sin embargo, al ejecutar el formulario se muestra el mensaje, pero el puntero de registro no se mueve porque nunca se ejecuta el código del evento Click de la clase primaria. Para asegurarse de que también se ejecuta el código del evento Click de la clase primaria, incluya las siguientes líneas de código en el procedimiento del evento Click del objeto:

```
DODEFAULT( )
WAIT WINDOW "En la parte inferior de la tabla" TIMEOUT 1
```

Nota Puede utilizar la función ACLASS( ) para determinar todas las clases de la jerarquía de clases de un objeto.

Impedir la ejecución del código de clase de base

En algunos casos, deseará evitar que produzca el comportamiento predeterminado de la clase de base en un evento o método. Para ello, incluya la palabra clave NODEFAULT en el código de método que escriba. Por ejemplo, el programa siguiente utiliza la palabra clave NODEFAULT en el evento

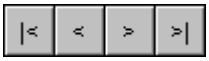
KeyPress de un cuadro de texto para impedir que se muestren en el cuadro los caracteres escritos:

```
frmKeyExample = CREATEOBJECT("prueba")
frmKeyExample.Show
READ EVENTS
DEFINE CLASS prueba AS FORM
  ADD OBJECT texto1 AS TEXTBOX
  PROCEDURE texto1.KeyPress
     PARAMETERS nKeyCode, nShiftAltCtrl
     NODEFAULT
     IF BETWEEN(nKeyCode, 65, 122) && entre 'A' y 'z'
        This.Value = ALLTRIM(This.Value) + "*"
        ACTIVATE SCREEN      && enviar el resultado a la ventana principal de Visua
        ?? CHR(nKeyCode)
     ENDIF
  ENDPROC
  PROCEDURE Destroy
     CLEAR EVENTS
  ENDPROC
ENDDEFINE
```

Crear un conjunto de botones de desplazamiento por tablas

Una característica común de muchas aplicaciones es una serie de botones de desplazamiento que permiten a los usuarios moverse por una tabla. Suelen incluir botones para mover el puntero de registro al registro siguiente o anterior de la tabla, así como al registro superior o inferior de la tabla.

Botones de desplazamiento por tablas



Diseño de los botones de desplazamiento

Todos los botones tendrán algunas características y funciones comunes, por lo que es conveniente crear una clase de botones de desplazamiento. A continuación, los botones individuales pueden aprovechar fácilmente esta apariencia y funcionalidad comunes. Esta clase primaria es la clase `NavButton` que se definirá posteriormente en esta sección.

Una vez definida la clase primaria, las subclases siguientes definen la funcionalidad y apariencia específicas de cada uno de los cuatro botones de desplazamiento: `navTop, navPrior, navNext, navBottom.`

Por último se crea una clase de contenedor `vcr`, a la que se agregan todos los botones de desplazamiento. El contenedor puede agregarse a un formulario o una barra de herramientas para proporcionar funcionalidad de desplazamiento por tablas.

Definición de la clase NAVBUTTON

Para crear `NavButton`, guarde las seis definiciones de clase siguientes (`Navbutton, navTop, navBottom, navPrior, navNext` y `vcr`) en un archivo de programa como Navclass.prg.

Definición de la clase genérica botón de comando de desplazamiento

Código	Comentarios
`DEFINE CLASS NavButton AS COMMANDBUTTON`	Define la clase primaria de los botones de desplazamiento.

`Height = 25`
`Width = 25`
`TableAlias = ""`
	Asigna dimensiones a la clase.
	
	Incluye una propiedad personalizada, `TableAlias`, que contiene el nombre del alias por el que desplazarse.

`PROCEDURE Click`
`IF NOT EMPTY(This.TableAlias)`
`SELECT (This.TableAlias)`
`ENDIF`
`ENDPROC`
	Si se ha establecido `TableAlias`, este procedimiento de clase priamria selecciona el alias antes de ejecutar el código real de desplazamiento en las subclases. De lo contrario, se supondrá que el usuario desea desplazarse por la tabla del área de trabajo seleccionada actualmente.

`PROCEDURE RefreshForm`
`_SCREEN.ActiveForm.Refresh`
`ENDPROC`
	Al emplear _SCREEN.ActiveForm.Refresh en lugar de THISFORM.Refresh puede agregar la clase a un formulario o una barra de herramientas y hacer que funcione con la misma precisión.

`ENDDEFINE`	Finaliza la definición de clase.

Los botones de desplazamiento específicos se basan en la clase `NavButton`. El código siguiente define el botón Superior para el conjunto de botones de desplazamiento. Los tres botones de desplazamiento restantes se definen en la tabla siguiente. Las cuatro definiciones de clase son similares. Por ello, sólo se ofrecen comentarios extensos para la primera definición.

Definición de la clase botón de desplazamiento Superior

Código	Comentarios
`DEFINE CLASS navTop AS BotDespl` `Caption = "|<"`	Define la clase botón de desplazamiento Superior y establece la propiedad Caption.
`PROCEDURE Click`	Crea código de método que se ejecutará cuando se produzca el evento Click para el control.

`DODEFAULT( )`
 	
`GO TOP`
 	
`THIS.RefreshForm`
	Llama al código de evento Click de la clase primaria, `Navbutton`, de modo que se pueda seleccionar el alias adecuado si se ha establecido la propiedad TableAlias.
	
	Incluye el código para establecer el puntero de registro en el primer registro de la tabla: GO TOP.

	Llama al método RefreshForm de la clase primaria. No es necesario utilizar el operador de resolución de alcance (::) en este caso porque no hay ningún método en la subclase que tenga el mismo nombre que el método de la clase primaria. Por otra parte, tanto la clase primaria como la subclase tienen código de método para el evento Click.
`ENDPROC`	Termina el procedimiento Click.
`ENDDEFINE`	Termina la definición de clase.

Los restantes botones de desplazamiento tienen definiciones de clase similares.

Definición de las demás clases de botones de desplazamiento

Código	Comentarios
`DEFINE CLASS navNext AS Navbutton` `  Caption = ">"`	Define la clase de botón de desplazamiento Siguiente y establece la propiedad Caption.

`PROCEDURE Click`
`  DODEFAULT( )`
`  SKIP 1`
`  IF EOF( )`
`  GO BOTTOM`
`  ENDIF`
`  THIS.RefreshForm`
`ENDPROC`
`ENDDEFINE`
	Incluye el código para establecer el puntero de registro en el siguiente registro de la tabla.
	
	Termina la definición de la clase.

`DEFINE CLASS navPrior AS Navbutton` `  Caption = "<"`	Define la clase de botón de desplazamiento Anterior y establece la propiedad Caption.

`PROCEDURE Click`
`  DODEFAULT( )`
`  SKIP -1`
`  IF BOF( )`
`  GO TOP`
`  ENDIF`
`  THIS.RefreshForm`
`ENDPROC`
`ENDDEFINE`
	Incluye el código para establecer el puntero de registro en el registro anterior de la tabla.
	
	Termina la definición de clase.

`DEFINE CLASS navBottom AS Navbutton` `  Caption = ">|"`	Define la clase de botón de desplazamiento Inferior y establece la propiedad Caption.

`PROCEDURE Click`
`  DODEFAULT( )`
`  GO BOTTOM`
`  THIS.RefreshForm`
`ENDPROC`
`ENDDEFINE`
	Incluye el código para establecer el puntero de registro en el último registro de la tabla.
	
	Termina la definición de clase.

La siguiente definición de clase contiene los cuatro botones de desplazamiento para poder agregarlos como una unidad a un formulario. La clase también incluye un método para establecer la propiedad TableAlias de los botones.

Definición de una clase de controles de desplazamiento por tabla

Código	Comentarios

`DEFINE CLASS vcr AS CONTAINER`
`  Height = 25`
`  Width = 100`
`  Left = 3`
`  Top = 3`
	Comienza la definición de clase. La propiedad Height se establece en el mismo alto que los botones de comando que contendrá.

`  ADD OBJECT cmdTop AS navTop ;`
`  WITH Left = 0`
`  ADD OBJECT cmdPrior AS navPrior ;`
`  WITH Left = 25`
`  ADD OBJECT cmdNext AS navNext ;`
`  WITH Left = 50`
`  ADD OBJECT cmdBot AS navBottom ;`
`  WITH Left = 75`
	Agrega los botones de desplazamiento.

`PROCEDURE SetTable(cTableAlias)`
`  IF TYPE("cTableAlias") = 'C'`
`  THIS.cmdTop.TableAlias = ;`
`  cTableAlias`
`  THIS.cmdPrior.TableAlias = ;`
`  cTableAlias`
`  THIS.cmdNext.TableAlias = ;`
`  cTableAlias`
`  THIS.cmdBot.TableAlias = ;`
`  cTableAlias`
`  ENDIF`
`ENDPROC`
	Este método se utiliza para establecer la propiedad TableAlias de los botones. TableAlias se define en la clase primaria `Navbutton`.
	
	También podría utilizar el método SetAll para establecer esta propiedad:
	`IF TYPE ("cTableAlias") = 'C'`
	`This.SetAll("TableAlias", "cTableAlias")`
	`ENDIF`
	Sin embargo, esto produciría un error si se agregara a la clase un objeto que no tuviera la propiedad TableAlias.

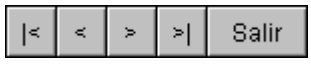
`ENDDEFINE`	Termina la definición de clase.

Una vez definida la clase, puede dividirla en subclases o agregarla a un formulario.

Crear una subclase basada en la nueva clase

También puede crear subclases basadas en `vcr` que tengan botones adicionales como Buscar, Modificar, Guardar y Salir. Por ejemplo, `vcr2` incluye un botón Salir:

Botones de desplazamiento por tablas con un botón para cerrar el formulario



Definición de una subclase de control de desplazamiento por tablas

Código	Comentarios

`DEFINE CLASS vcr2 AS vcr`
`ADD OBJECT cmdQuit AS`
`COMMANDBUTTON WITH ;`
`  Caption = "Salir",;`
`  Height = 25, ;`
`  Width = 50`
`Width = THIS.Width + THIS.cmdQuit.Width`
`cmdQuit.Left = THIS.Width - ;`
`  THIS.cmdQuit.Width`
	Define una clase basada en `vcr` y le agrega un botón de comando.

`PROCEDURE cmdQuit.CLICK`
`  RELEASE THISFORM`
`ENDPROC`
	Cuando el usuario haga clic en `cmdQuit`, este código liberará el formulario.

`ENDDEFINE`	Termina la definición de clase.

`Vcr2` tiene todo lo de `vcr` más el nuevo botón de comando y no es necesario volver a escribir ninguna parte del código.

Cambios en VCR reflejados en la subclase

A causa de la herencia, los cambios realizados en la clase primaria se reflejan en todas las subclases que se basan en ella. Por ejemplo, puede informar al usuario de que se ha llegado al final de la tabla si cambia la instrucción `IF EOF( )` de `navNext.Click` por la siguiente:

```
IF EOF( )
   GO BOTTOM
   SET MESSAGE TO "Final de la tabla"
ELSE
   SET MESSAGE TO
ENDIF
```

Puede indicar al usuario que ha llegado al principio de la tabla si cambia la instrucción `IF BOF( )` de `navPrior.Click` por la siguiente:

```
IF BOF()
```

```
   GO TOP
   SET MESSAGE TO "Principio de la tabla"
ELSE
   SET MESSAGE TO
ENDIF
```

Si se realizan estos cambios en las clases `navNext` y `navPrior`, también se aplicarán automáticamente a los botones apropiados de `vcr` y `vcr2`.

Agregar vcr a una clase de formulario

Una vez definido `vcr` como un control, el control puede agregarse a la definición de un contenedor. Por ejemplo, el código siguiente agregado a Navclass.prg define un formulario al que se han agregado botones de desplazamiento:

```
DEFINE CLASS NavForm AS Form
   ADD OBJECT oVCR AS vcr
ENDDEFINE
```

Ejecutar el formulario que contiene VCR

Una vez definida la subclase de formulario, podrá mostrarla fácilmente con los comandos apropiados.

Para mostrar el formulario

1. Cargue la definición de clase:

   ```
   SET PROCEDURE TO navclass ADDITIVE
   ```

2. Cree un objeto basado en la clase `navForm`:

   ```
   frmPrueba = CREATEOBJECT("navForm")
   ```

3. Invoque el método Show del formulario:

   ```
   frmPrueba.Show
   ```

Si no llama al método SetTable de `oVCR` (el objeto VCR de `NavForm`), cuando el usuario haga clic en los botones de desplazamiento el puntero de registro se moverá por la tabla del área de trabajo seleccionada actualmente. Puede llamar al método SetTable para especificar en qué tabla se va a desplazar.

```
frmPrueba.oVCR.SetTable("customer")
```

Nota Cuando el usuario cierre el formulario, `frmPrueba` se establecerá a un valor nulo (.NULL.). Para liberar de la memoria la variable de objeto, utilice el comando RELEASE. Las variables de objeto creadas en los archivos de programa se liberan de la memoria cuando se completa el programa.

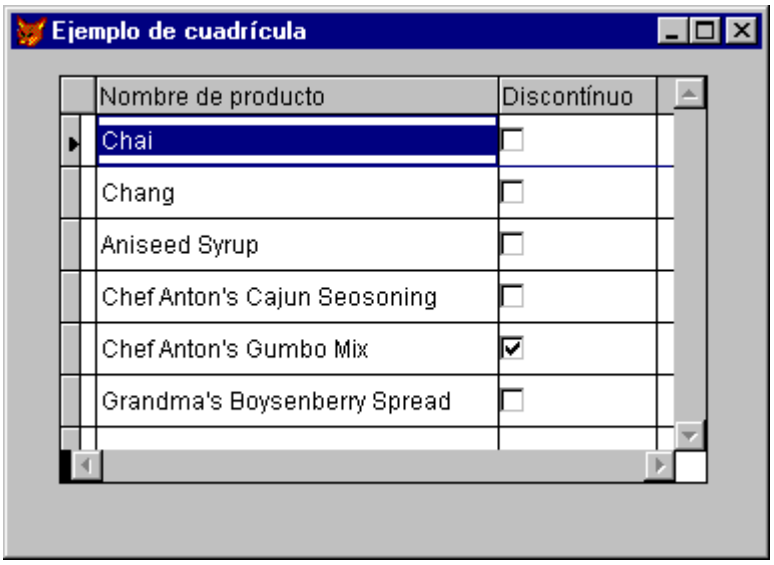
Definir un control cuadrícula

Una cuadrícula contiene columnas que, a su vez, pueden contener encabezados y cualquier otro

control. El control predeterminado contenido en una columna es un cuadro de texto, por lo que la funcionalidad predeterminada de la cuadrícula se aproxima a una ventana Examinar. Sin embargo, la arquitectura subyacente de la cuadrícula la abre hasta una extensión ilimitada.

El ejemplo siguiente crea un formulario que contiene un objeto Grid (Cuadrícula) con dos columnas. La segunda columna contiene una casilla de verificación para mostrar los valores en un campo lógico de una tabla.

Control Grid (Cuadrícula) con una casilla de verificación en una columna



Definición de una clase Grid con una casilla de verificación en una columna de cuadrícula

Código	Comentarios

`DEFINE CLASS grdProducts AS Grid`
`  Left = 24`
`  Top = 10`
`  Width = 295`
`  Height = 210`
`  Visible = .T.`
`  RowHeight = 28`
`  ColumnCount = 2`
	Comienza la definición de clase y establece las propiedades que determinan la apariencia de la cuadrícula.
	
	Al establecer la propiedad ColumnCount en 2, se agregan dos columnas a la cuadrícula. Cada columna contiene un encabezado con el nombre Header1. Además, cada columna tiene un grupo de propiedades independiente que determina su apariencia y comportamiento.

`Column1.ControlSource ="prod_name"` `Column2.ControlSource ="discontinu"`	Al establecer la propiedad ControlSource de una columna, la columna muestra los valores de ese campo para todos los registros de la tabla. `Discontinu` es un campo lógico.
`Column2.Sparse = .F.`	Column2 contendrá la casilla de verificación. Establezca la propiedad Sparse de la columna

	en .F. de modo que la casilla de verificación sea visible en todas las filas, no sólo en la celda seleccionada.

`Procedure Init`
`  THIS.Column1.Width = 175`
`  THIS.Column2.Width = 68`
`  THIS.Column1.Header1.Caption = ;`
`  "Nombre de producto"`
`  THIS.Column2.Header1.Caption = ;`
`  "Suspendido"`
 	
`  THIS.Column2.AddObject("chk1", ;`
`  "checkbox")`
`  THIS.Column2.CurrentControl = ;`
`  "chk1"`
`  THIS.Column2.chk1.Visible = .T.`
`  THIS.Column2.chk1.Caption = ""`
`ENDPROC`
	Establece el ancho de las columnas y los títulos de los encabezados.
	
	El método AddObject permite agregar un objeto a un contenedor ; en este caso, una casilla de verificación llamada `chk1`.
	Establece la propiedad CurrentControl de la columna en la casilla de verificación, de modo que se muestre la casilla de verificación.
	Comprueba que la casilla de verificación es visible.
	Establece el título en una cadena vacía de modo que no se muestre el título predeterminado "chk1".

`ENDDEFINE`	Termina la definición de clase.

La siguiente definición de clase es el formulario que contiene la cuadrícula. Ambas definiciones de clase pueden incluirse en el mismo archivo de programa.

Definición de una clase Form que contiene la clase Grid

Código	Comentarios

`DEFINE CLASS GridForm AS FORM`
`  Width = 330`
`  Height = 250`
`  Caption = "Ejemplo de cuadrícula"`
`  ADD OBJECT grid1 AS grdProducts`
	Crea una clase de formulario y le agrega un objeto basado en la clase de cuadrícula.

`PROCEDURE Destroy`
`  CLEAR EVENTS`
`ENDPROC`
 	
`ENDDEFINE`
	El programa que crea un objeto basado en esta clase utilizará READ EVENTS. Al incluir CLEAR EVENTS en el evento Destroy del formulario, el programa podrá terminar cuando el usuario cierre el formulario.
	Termina la definición de clase.

El programa siguiente abre la tabla donde están incluidos los campos que se van a mostrar en las columnas de cuadrícula, crea un objeto basado en la clase GridForm y ejecuta el comando READ EVENTS.

```
CLOSE DATABASE
OPEN DATABASE (SYS(2004) + "samples\data\testdata.dbc")
USE products
frmTest= CREATEOBJECT("GridForm")
frmTest.Show
READ EVENTS
```

Este programa puede incluirse en el mismo archivo en el que están incluidas las definiciones de clase si aparece al principio del archivo. También puede emplear el comando SET PROCEDURE TO para especificar el programa que contiene las definiciones de clase e incluir este código en un programa distinto.

Crear referencias a objetos

En lugar de realizar una copia de un objeto, puede crear una referencia a dicho objeto. Una referencia ocupa menos memoria que un objeto adicional, puede transferirse fácilmente entre procedimientos y puede ayudar a escribir código genérico.

Devolver una referencia a un objeto

En algunas ocasiones puede resultar conveniente manipular un objeto por medio de una o varias referencias al mismo. Por ejemplo, el programa siguiente define una clase, crea un objeto basado en la clase y devuelve una referencia al objeto:

```
*--NEWINV.PRG
*--Devuelve una referencia a un nuevo formulario de facturas.
frmInv = CREATEOBJECT("InvoiceForm")
RETURN frmInvoice

DEFINE CLASS InvoiceForm AS FORM
   ADD OBJECT txtCompany AS TEXTBOX
   * código para establecer propiedades, agregar otros objetos, etc.
ENDDEFINE
```

El programa siguiente establece una referencia al objeto creado en Newin.prg. La variable de referencia puede manipularse exactamente del mismo modo que la variable de objeto:

```
frmInvoice = NewInv() && almacena la referencia al objeto en una variable
frmInvoice.SHOW
```

También puede crear una referencia a un objeto de un formulario, como en el ejemplo siguiente.

```
txtCustName = frmInvoice.txtCompany
txtCustName.Value = "Usuario de Fox"
```

Sugerencia Cuando ha creado un objeto, puede usar el comando DISPLAY OBJECTS para mostrar la jerarquía de clases del objeto, los valores de las propiedades, los objetos contenidos y los métodos y eventos disponibles. Puede llenar una matriz con las propiedades (no los valores de las propiedades), eventos, métodos y objetos contenidos de un objeto con la función AMEMBERS( ).

Liberar objetos y referencias de la memoria

Si existe una referencia a un objeto, la liberación del objeto no borra el objeto de la memoria. Por ejemplo, el comando siguiente libera el objeto original, `frmFactura`:

```
RELEASE frmFactura
```

Sin embargo, puesto que sigue existiendo una referencia a un objeto perteneciente a `frmFactura`, el objeto no se liberará de la memoria hasta que se libere `txtNombrePers` con el comando siguiente:

```
RELEASE txtNombrePers
```

Comprobar si existe un objeto

Puede utilizar las funciones TYPE( ), ISNULL( ) y VARTYPE( ) para determinar si existe un objeto. Por ejemplo, las líneas de código siguientes comprueban si existe un objeto llamado `oConexión`:

```
IF TYPE("oConexión") = "O" AND NOT ISNULL(oConexión)
   * El objeto existe
ELSE
   * El objeto no existe
ENDIF
```

Nota El comando ISNULL( ) es necesario porque .NULL. se almacena en la variable de objeto de formulario cuando un usuario cierra un formulario, pero el tipo de variable sigue siendo "O".

Crear matrices de miembros

Puede definir miembros de clases como matrices. En el ejemplo siguiente, `elecc` es una matriz de controles.

```
DEFINE CLASS MoverListBox AS CONTAINER
DIMENSION choices[3]
ADD OBJECT lFromListBox AS LISTBOX
ADD OBJECT lToListBox AS LISTBOX
ADD OBJECT choices[1] AS COMMANDBUTTON
ADD OBJECT choices[2] AS COMMANDBUTTON
ADD OBJECT choices[3] AS CHECKBOX
PROCEDURE choices.CLICK
   PARAMETER nIndex
   DO CASE
      CASE nIndex = 1
         * código
      CASE nIndex = 2
         * código
      CASE nIndex = 3
         * código
   ENDCASE
ENDPROC
ENDDEFINE
```

Cuando un usuario hace clic en un control incluido en una matriz de controles, Visual FoxPro transfiere el número de índice del control al procedimiento de evento Click. En este procedimiento,

puede utilizar una instrucción CASE para ejecutar código distinto según el botón en el que se haya hecho clic.

Crear matrices de objetos

También puede crear matrices de objetos. Por ejemplo, `MiMatriz` contiene cinco botones de comando:

```
DIMENSION MiMatriz[5]
FOR x = 1 TO 5
   MiMatriz[x] = CREATEOBJECT("COMMANDBUTTON")
ENDFOR
```

Hay una serie de consideraciones que conviene tener en cuenta con respecto a las matrices de objetos:

- No se puede asignar un objeto a una matriz completa mediante un comando. Es necesario asignar individualmente el objeto a cada miembro de la matriz.
- No se puede asignar un valor a una propiedad de una matriz completa. El comando siguiente produciría un error:

  ```
  MiMatriz.Enabled = .F.
  ```

- Al redimensionar una matriz de objetos para que sea más grande que la matriz original, los elementos nuevos se inicializarán como falso (.F.), como ocurre con todas las matrices de Visual FoxPro. Cuando redimensione una matriz de objetos para que sea más pequeña que la matriz original, se liberarán los objetos cuyo subíndice sea mayor que el mayor subíndice nuevo.

Usar objetos para almacenar datos

En los lenguajes orientados a objetos, una clase ofrece un medio útil y cómodo para almacenar datos y procedimientos relacionados con una entidad. Por ejemplo, podría definir una clase de cliente para incluir en ella información sobre un cliente, así como un método para calcular la edad del cliente:

```
DEFINE CLASS cliente AS CUSTOM
   Apellidos = ""
   Nombre = ""
   FechaNacimiento = { - - }
   PROCEDURE Edad
      IF !EMPTY(THIS.FechaNacimiento)
         RETURN YEAR(DATE()) - YEAR(THIS.FechaNacimiento)
      ELSE
         RETURN 0
      ENDIF
   ENDPROC
ENDDEFINE
```

Sin embargo, los datos almacenados en objetos que se basan en la clase de cliente sólo se almacenan en memoria. Si estos datos estuvieran en una tabla, ésta se almacenaría en disco. Si tuviera que hacer un seguimiento de varios clientes, la tabla le daría acceso a todos los comandos y las funciones de administración de bases de datos de Visual FoxPro. De este modo, podría localizar información rápidamente, ordenarla, agruparla, realizar cálculos, crear informes y consultas basándose en la

información, etc.

Visual FoxPro ofrece un resultado incomparable en cuanto al almacenamiento y la manipulación de datos de bases de datos y tablas. Sin embargo, en determinadas ocasiones deseará almacenar datos en objetos. Generalmente, los datos sólo serán significativos mientras se esté ejecutando la aplicación y pertenecerán a una única entidad.

Por ejemplo, en una aplicación que incluye un sistema de seguridad, normalmente tendría una tabla de los usuarios que tienen acceso a la aplicación. La tabla incluiría la identificación, la contraseña y el nivel de acceso del usuario. Cuando un usuario haya iniciado una sesión no necesitará toda la información de la tabla. Lo único que necesitará es la información sobre el usuario actual y esta información se puede almacenar y manipular fácilmente en un objeto. Por ejemplo, la definición de clase siguiente inicia una sesión al crear un objeto basado en la clase:

```
DEFINE CLASS NuevoUsuario AS CUSTOM
   PROTECTED LogonTime, AccessLevel
   UserId = ""
   PassWord = ""
   LogonTime = { - - : : }
   AccessLevel = 0
   PROCEDURE Init
      DO FORM LOGON WITH ; && suponiendo que ha creado este formulario
         This.UserId, ;
         This.PassWord, ;
         This.AccessLevel
      This.LogonTime = DATETIME( )
   ENDPROC
* Crear métodos para devolver valores de propiedad protegidos.
   PROCEDURE GetLogonTime
      RETURN This.LogonTime
   ENDPROC
   PROCEDURE GetAccessLevel
      RETURN This.AccessLevel
   ENDPROC
ENDDEFINE
```

En el programa principal de la aplicación, podría crear un objeto basado en la clase `NuevoUsuario`:

```
oUser = CREATEOBJECT('NuevoUsuario')
oUser.Logon
```

En cualquier parte de la aplicación, cuando necesite información sobre el usuario actual, podrá obtenerla del objeto `oUser`. Por ejemplo:

```
IF oUser.GetAccessLevel( ) >= 4
   DO ADMIN.MPR
ENDIF
```

Integrar objetos y datos

En la mayoría de las aplicaciones, puede sacar el máximo partido de la potencia de Visual FoxPro si integra objetos y datos. La mayoría de las clases de Visual FoxPro tienen propiedades y métodos que permiten integrar la potencia de un administrador de base de datos relacional y un sistema completamente orientado a objetos.

Propiedades para integrar datos de clases y bases de datos de Visual FoxPro

Clase	Propiedades de datos
Cuadrícula	RecordSource, ChildOrder, LinkMaster
Todos los demás controles	ControlSource
Cuadro de lista y cuadro combinado	ControlSource, RowSource
Formulario y conjunto de formularios	DataSession

Puesto que estas propiedades de datos pueden cambiarse en tiempo de diseño o en tiempo de ejecución, puede crear controles genéricos con funcionalidad encapsulada que opere con datos diversos.

Para obtener más información sobre la integración de datos y objetos, consulte el capítulo 9, Crear formularios y el capítulo 10, Usar controles.

Capítulo 4: Descripción del modelo de eventos

Visual FoxPro ofrece un auténtico funcionamiento no modal, por lo que es posible coordinar fácilmente múltiples formularios automáticamente y ejecutar simultáneamente múltiples instancias de un formulario. Además, Visual FoxPro se encarga del procesamiento de los eventos, por lo que puede ofrecer a sus usuarios un entorno interactivo mucho más rico.

En este capítulo se describe:

- Eventos de Visual FoxPro
- Seguimiento de secuencias de eventos
- Asignar código a eventos

Eventos de Visual FoxPro

El sistema desencadena automáticamente un código de evento como respuesta a alguna acción del usuario. Por ejemplo, el sistema procesa automáticamente el código escrito para el evento Click cuando el usuario hace clic en un control. El código de un evento también puede desencadenarse mediante eventos del sistema, como es el caso del evento Timer en un control de cronómetro.

Los eventos básicos

La tabla siguiente contiene una lista del principal conjunto de eventos de Visual FoxPro que se aplican a la mayoría de los controles.

Conjunto básico de eventos

Evento	Cuándo se desencadena el evento
Init	Al crear un objeto.
Destroy	Al liberar de la memoria un objeto.
Click	Cuando el usuario hace clic en el objeto con el botón principal del *mouse*.
DblClick	Cuando el usuario hace doble clic en el objeto con el botón principal del *mouse*.
RightClick	Cuando el usuario hace clic en el objeto con el botón secundario del *mouse*.
GotFocus	Cuando el objeto recibe el enfoque, ya sea como resultado de una acción del usuario o al hacer clic, o porque se cambie el enfoque en el código mediante el método SetFocus.
LostFocus	Cuando el objeto pierde el enfoque, ya sea como resultado de una acción del usuario o al hacer clic, o porque se cambie el enfoque en el código mediante el método SetFocus.
KeyPress	Cuando el usuario presiona y suelta una tecla.
MouseDown	Cuando el usuario presiona el botón del *mouse* mientras el puntero del *mouse* se encuentra sobre el objeto.
MouseMove	Cuando el usuario mueve el *mouse* sobre el objeto.
MouseUp	Cuando el usuario libera un botón del *mouse* mientras el puntero del *mouse* se encuentra sobre el objeto.

Contenedores y eventos de objeto

A la hora de escribir código de eventos para los controles se deben tener en cuenta dos reglas básicas:

- Los contenedores no procesan los eventos asociados a los controles que contienen.
- Si no hay código de evento asociado a un control, Visual FoxPro comprobará si hay código asociado al evento en algún nivel superior de la jerarquía de clase para dicho control.

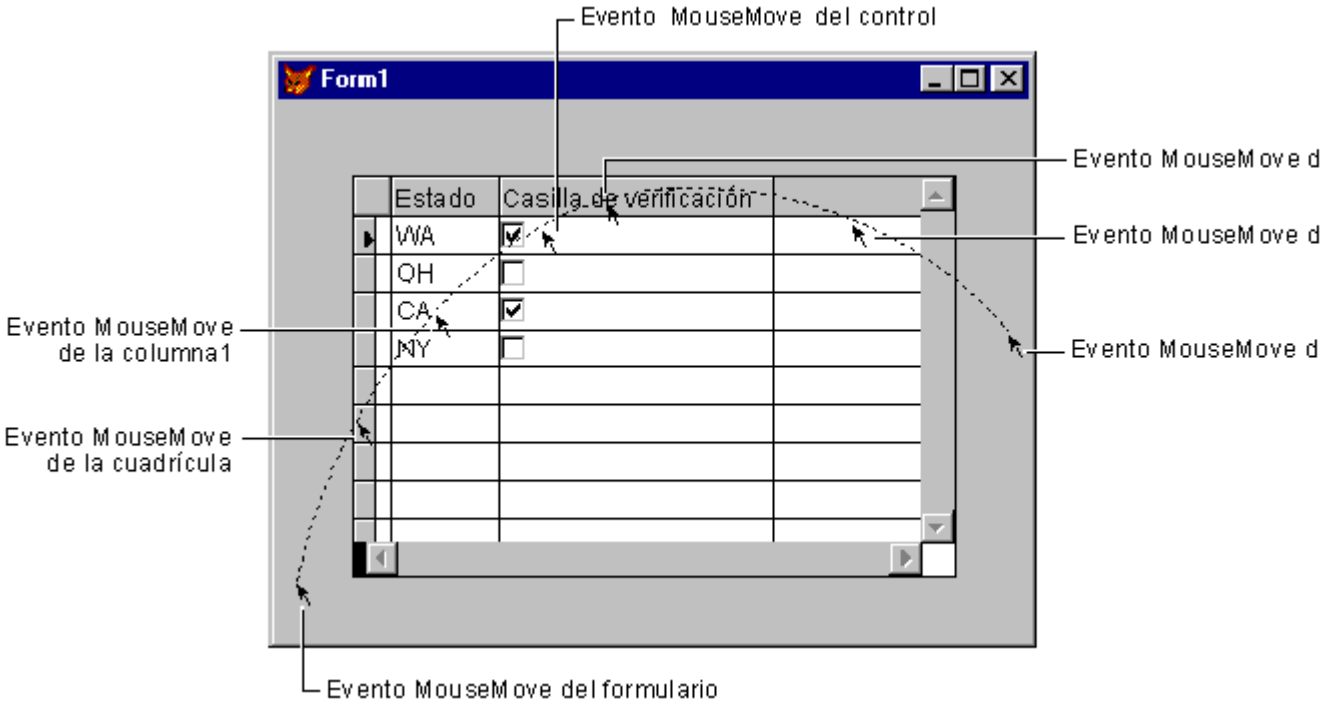
Cuando un usuario interactúa con un objeto de alguna forma, ya sea presionando la tecla tab, haciendo clic en él, moviendo el puntero del *mouse* sobre él, etc., tienen lugar los eventos de objeto. Cada objeto recibe sus eventos de forma independiente. Por ejemplo, aunque un botón de comando se encuentre en un formulario, el evento Click del formulario no se desencadenará cuando un usuario haga clic en el botón de comando; sólo se desencadenará el evento Click del botón de comando.

El código de evento del contenedor es distinto del código de evento del control

Si no hay ningún código de evento Click asociado al botón de comando, cuando el usuario haga clic en el botón no ocurrirá nada, incluso cuando haya un código de evento Click asociado al formulario.

Esta regla también es aplicable a los controles cuadrícula. La cuadrícula contiene columnas que a su vez contienen encabezados y controles. Cuando ocurren los eventos, sólo el objeto más interno implicado en el evento reconoce el evento. Los contenedores de mayor nivel no reconocen el evento. La ilustración siguiente muestra qué objetos procesan los eventos MouseMove que se generan cuando un usuario mueve el puntero del *mouse* por la cuadrícula.

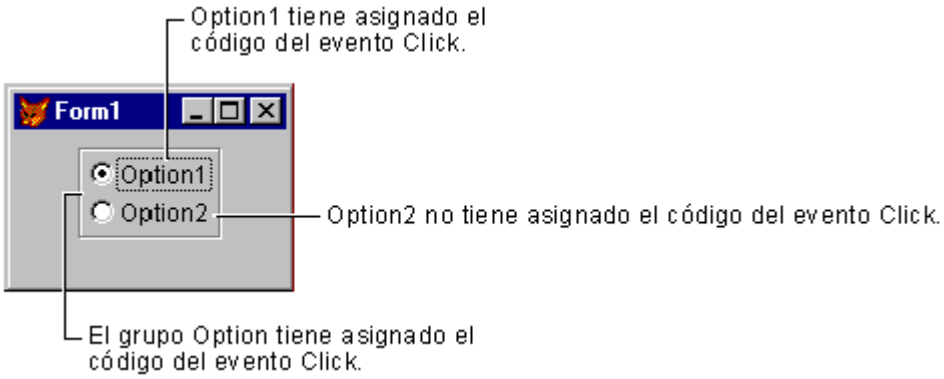
Eventos MouseMove para una cuadrícula



No obstante, hay una excepción a esta regla. Si ha escrito código de evento para un grupo de botones de opción o para un grupo de botones de comando pero no hay código para el evento en un determinado botón del grupo, el código de evento del grupo *se* ejecutará cuando se produzca el evento del botón.

Por ejemplo, podría tener un grupo de botones de opción con un código de evento Click asociado. Sólo uno de los dos botones de opción del grupo tienen asociado código de evento Click:

El código de evento para los grupos de botones puede utilizarse como valor predeterminado



Si un usuario hace clic en Opción1, se ejecutará el código de evento Click asociado a Opción1. El código de evento Click asociado al grupo de botones de opción no se ejecutará.

Puesto que no hay código de evento Click asociado a Opción2, si el usuario hace clic en Opción2 se ejecutará el código de evento Click del grupo de opciones.

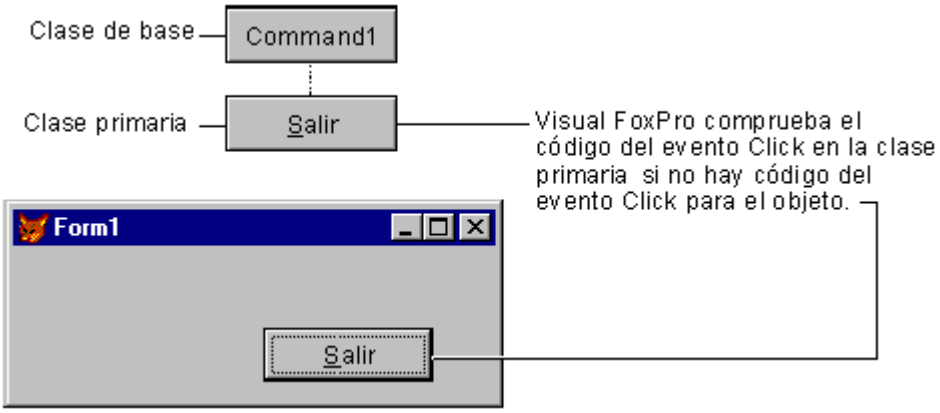
Nota Cuando se inicia una secuencia de eventos, como MouseDown y MouseUp, para un control, toda la secuencia de eventos pertenece al control.

Por ejemplo, si presiona el botón primario del *mouse* en un botón de comando y arrastra el puntero del *mouse* hacia fuera del botón de comando, los eventos MouseMove del botón de comando seguirán produciéndose, aunque el puntero del *mouse* se esté moviendo fuera del formulario. Si suelta el botón primario del *mouse* sobre el formulario en lugar de hacerlo sobre el botón de comando, el evento MouseUp que ocurrirá será el asociado al botón de comando, no al formulario.

Clases y eventos de controles

Si un control de un formulario está basado en una clase definida por el usuario (que, a su vez, está basada en otra clase definida por el usuario), cuando se produzca un evento, Visual FoxPro comprobará el código de evento del control inmediato. Si hay código en ese procedimiento de evento, Visual FoxPro lo ejecutará. Si no existe código en el procedimiento de evento, Visual FoxPro comprobará un nivel superior en la jerarquía de clases. Si en algún lugar de la jerarquía de clases Visual FoxPro encuentra código para el evento, se ejecutará dicho código. Cualquier código que haya más allá dentro de la jerarquía no se ejecutará.

Si no hay código de evento asociado a un objeto, Visual FoxPro comprobará la clase primaria.



No obstante, puede incluir código en un procedimiento de evento y llamar explícitamente al código en clases en las cuales se base el control; para ello se utiliza la función DODEFAULT( ).

Seguimiento de secuencias de eventos

El modelo de eventos de Visual FoxPro es amplio y le concede bastante control sobre los componentes de la aplicación en respuesta a una amplia variedad de acciones de usuario. Algunas de las secuencias de eventos son fijas como, por ejemplo, la creación o destrucción de un formulario. Algunos eventos ocurren de forma independiente, pero la mayor parte ocurre con otros eventos basados en la interacción con el usuario.

Establecer el seguimiento de eventos

La mejor manera de ver las secuencias de eventos de Visual FoxPro es establecer el seguimiento de eventos en el depurador. El seguimiento de eventos le permite ver cuándo tiene lugar cada evento asociado a sus propios formularios y controles en relación a otros eventos, de forma que puede determinar el lugar más eficiente para incluir el código.

Para establecer el seguimiento de eventos

1. En el menú **Herramientas** de la ventana Depurador, elija **Seguimiento de eventos**.

2. En el cuadro de diálogo **Seguimiento de eventos**, seleccione **Activar el seguimiento de eventos**.

Los eventos de la lista Eventos para seguir se escriben en la ventana Salida del depurador o en un archivo cuando tengan lugar.

El cuadro de diálogo Seguimiento de eventos

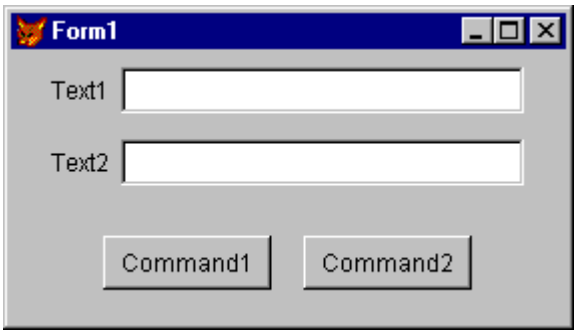
Nota En este ejemplo, los eventos MouseMove y Paint han sido eliminados de la lista Eventos para seguir porque ocurren con tanta frecuencia que hacen más difícil ver las secuencias de los otros eventos.

Observar cómo ocurren los eventos

A veces una acción de un usuario desencadena un único evento, como mover el puntero del *mouse* sobre un control, por ejemplo. Sin embargo, con frecuencia una acción del usuario desencadena múltiples eventos.

En esta sección se describe el orden en que ocurren los eventos como respuesta a la interacción del usuario, utilizando el siguiente formulario como ejemplo.

Formulario de ejemplo para ilustrar las secuencias de eventos



En esta situación de ejemplo, el usuario realiza las siguientes acciones en el formulario:

1. Ejecuta el formulario.

2. Escribe texto en Text1.
3. Selecciona el campo y lo copia al Portapapeles.
4. Va a Text2.
5. Pega el texto en Text2.
6. Cierra el formulario haciendo clic en Command2.

Estas acciones desencadenan uno o más eventos del sistema para cada objeto. En las tablas siguientes se describen los eventos desencadenados como respuesta a cada acción del usuario.

Acción 1

El usuario ejecuta el formulario escribiendo el siguiente comando en la ventana Comandos:

```
DO FORM form1 NAME frmObject
```

Visual FoxPro carga el formulario, inicializa cada objeto y después inicializa el formulario; el formulario se activa y el primer campo recibe el enfoque de entrada.

Objeto	Evento
DataEnvironment	BeforeOpenTables
Form1	Load
DataEnvironment	Init
Text1	Init
Text2	Init
Command1	Init
Command2	Init
Form1	Init
Form1	Activate
Form1	GotFocus
Text1	When
Text1	GotFocus

Acción 2

El usuario escribe **Test** en Text1. Cada pulsación de teclas genera dos eventos. El evento KeyPress

recibe 2 parámetros: la tecla presionada y el estado de las teclas MAYÚS, ALT y CTRL.

Objeto	Evento
Text1	KeyPress(84, 1) “T”
Text1	InteractiveChange
Text1	KeyPress(101, 0) “e”
Text1	InteractiveChange
Text1	KeyPress(115,0) “s”
Text1	InteractiveChange
Text1	KeyPress(116,0) “t”
Text1	InteractiveChange

Acción 3

El usuario hace doble clic en Text1 para seleccionar el texto y presiona CTRL+C para copiar el texto al Portapapeles. Los eventos Mouse y un evento Click acompañan al evento DblClick. Los eventos MouseMove y MouseDown reciben cuatro parámetros: qué botón, el estado de MAYÚS y las ubicaciones X e Y. Las ubicaciones X e Y son relativas al formulario y reflejan el modo de escala (por ejemplo, píxeles) del formulario. Sólo se presenta un evento MouseMove para cada control. En realidad, este evento se activaría probablemente media docena de veces o más.

Objeto	Evento
Form1	MouseMove(0, 0, 100, 35)
Text1	MouseMove(0,0,44,22)
Text1	MouseDown(1, 0, 44, 22)
Text1	MouseUp(1, 0, 44, 22)
Text1	Click
Text1	MouseDown(1, 0, 44, 22)
Text1	MouseUp(1, 0, 44, 22)
Text1	DblClick

Acción 4

El usuario presiona TAB para pasar a Text2.

Objeto	Evento
Text1	KeyPress(9, 0)
Text1	Valid
Text1	LostFocus
Text2	When
Text2	GotFocus

Acción 5

El usuario pega en Text2 el texto copiado al presionar CTRL+V.

Objeto	Evento
Text2	InteractiveChange

Acción 6

El usuario hace clic en Command2, que cierra el formulario.

Objeto	Evento
Form1	MouseMove
Command2	MouseMove
Text2	Valid
Command2	When
Text2	LostFocus
Command2	GotFocus
Command2	MouseDown(1, 0, 143, 128)
Command2	MouseUp(1, 0, 143, 128)
Command2	Click
Command2	Valid
Command2	When

Cuando se cierra el formulario y se libera el objeto se producen estos eventos adicionales, en orden inverso a los eventos de la Acción 1.

Objetos	Evento
Form1	Destroy
Command2	Destroy
Command1	Destroy
Text2	Destroy
Text1	Destroy
Form1	Unload
DataEnvironment	AfterCloseTables
DataEnvironment	Destroy

La secuencia de eventos de Visual FoxPro

La tabla siguiente muestra la secuencia de activación general de los eventos de Visual FoxPro. Se supone que la propiedad AutoOpenTables del entorno de datos está establecida a verdadero (.T.). Otros eventos pueden tener lugar en base a interacciones de usuario y a respuesta del sistema.

Objeto	Evento
DataEnvironment	BeforeOpenTables
FormSet	Load
Form	Load
Cursores DataEnvironment	Init
DataEnvironment	Init
Objects [1]	Init
Form	Init
FormSet	Init
FormSet	Activate
Form	Activate
Object1 [2]	When
Form	GotFocus
Object1	GotFocus
Object1	Message

Object1	Valid [3]
Object1	LostFocus
Object2 [3]	When
Object2	GotFocus
Object2	Message
Object2	Valid [4]
Object2	LostFocus
Form	QueryUnload
Form	Destroy
Object [5]	Destroy
Form	Unload
FormSet	Unload
DataEnvironment	AfterCloseTables
DataEnvironment	Destroy
Cursores DataEnvironment	Destroy

[1] Para cada objeto, desde el objeto más interno hasta el contenedor más externo
[2] Primer objeto según el orden de tabulación
[3] Siguiente objeto que va a recibir el enfoque
[4] Cuando el objeto pierde el enfoque
[5] Para cada objeto, desde el contenedor más externo hasta el objeto más interno

Asignar código a eventos

A menos que asocie código a un evento, no pasará nada cuando se produzca dicho evento. Casi nunca escribirá código para los eventos asociados a cualquier objeto de Visual FoxPro, pero querrá incorporar funcionalidad como respuesta a ciertos eventos clave de sus aplicaciones. Para agregar código que se va a ejecutar cuando se produzca un evento, utilice la ventana Propiedades del Diseñador de formularios.

La secuencia de eventos afecta a dónde debe situar el código. Tenga en cuenta las siguientes sugerencias:

- El evento Init de todos los controles del formulario se ejecuta antes que el evento Init del formulario, por lo que puede incluir código en el evento Init del formulario para manipular cualquier control del formulario. Por ejemplo, si va a agregar controles para columnas de cuadrícula, podrá hacerlo en el evento Init del formulario.

- Si quiere que se procese parte del código siempre que el valor de cuadro de lista, cuadro combinado o casilla de verificación cambia, asóciela al evento InteractiveChange. El evento Click puede no tener lugar o puede ser llamado incluso si el valor no ha cambiado.
- Cuando arrastra un control, los otros eventos de *mouse* se paran. Por ejemplo, los eventos MouseUp y MouseMove no tienen lugar durante una operación de arrastrar y colocar.
- Los eventos Valid y When devuelven un valor, verdadero (.T.) de forma predeterminada. Si devuelve falso (.F.) ó 0 desde el evento When, el control no puede tener el enfoque. Si devuelve falso (.F.) ó 0 desde el evento Valid, el enfoque no puede abandonar el control.

Para obtener más información acerca del uso del Diseñador de formularios, consulte el capítulo 9, Crear formularios. Para obtener información acerca de la codificación de clases y la forma de agregar código de eventos, consulte el capítulo 3, Programación orientada a objetos.