Comencemos a programar con

VBA - Access

Entrega 11

Gestión de errores

Eduardo Olaz

Errores

La gestión de los errores en procedimientos

A la hora de utilizar el código de un módulo, hay dos tiempos

- Tiempo de **Diseño**.
- Tiempo de **Ejecución**.

El **tiempo de diseño** transcurre mientras estamos modificando el contenido del código de un módulo, sea del tipo que sea, o cuando estamos cambiando las propiedades de controles, formularios o informes, en la llamada Vista Diseño, ya sea directamente ó mediante ejecución de código.

> *Esto a más de uno le sorprenderá:*
> *Access permite crear mediante código, formularios por ejemplo utilizando el método* `CreateForm`*, que devuelve una referencia a un nuevo formulario, también permite añadirle controles mediante el método* `CreateControl` *, e incluso asociarle un módulo, escribiendo dinámicamente todo su contenido. Para esto último tendríamos que crear una referencia al objeto* `Module` *del formulario, y para insertarle el código utilizar su método* `InsertText`*.*
> *De forma semejante existe el método* `CreateReport` *para la creación dinámica de informes.*
> *Si queremos usarlos posteriormente deberemos guardarlos, por ejemplo con el método* `Save` *del objeto* `DoCmd`*.*

El **tiempo de ejecución** transcurre cuando hemos creado una instancia de un objeto, formulario, informe, clase o hemos llamado a un procedimiento de un módulo estándar.

En el lenguaje “normal” podríamos decir que estamos en tiempo de ejecución cuando estamos “ejecutando” los objetos o el código de Access.

Errores en Tiempo de Diseño

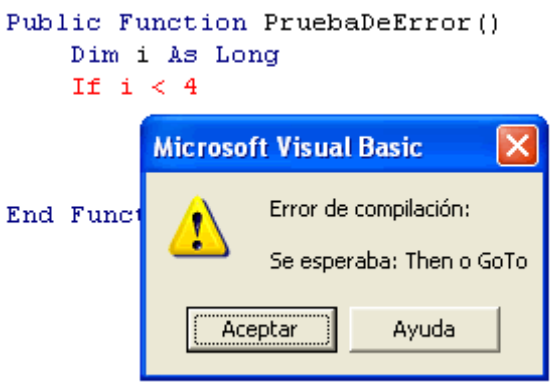
En Tiempo de Diseño podemos cometer una serie de errores, a la hora de escribir el código.

Muchos de estos errores serán detectados inmediatamente por el editor de Access.

Cuando escribimos una línea de código, Access realiza un análisis del texto que estamos escribiendo. En este proceso se realiza fundamentalmente su análisis sintáctico.

También comprueba si hay sentencias incompletas, por ejemplo `If` sin `Then`.

Si encuentra una expresión errónea lanza un mensaje de **Error de compilación** e incluso aporta una cierta información que nos puede orientar sobre el origen del error.



La línea de código incorrecta queda marcada en color rojo.

Cuando ejecutamos el código, la primera vez que lo hace, no sólo realiza un análisis sintáctico, además va comprobando que todas las constantes y variables, ya sean de tipos estándar, referencias de objetos ó tipos definidos por el usuario, estén perfectamente declaradas, y los tipos de objeto existan y sean correctos.

```
Public Function PruebaDeError()
    Dim i As Long
    n = 4
End Function
```

Si se detecta algún error se interrumpe la ejecución del código y se lanza un aviso, marcando la zona del código donde el error se ha producido. Esta depuración del código se va realizando conforme se efectúan llamadas a los diferentes procedimientos.

Podría ocurrir que tuviéramos un procedimiento que sólo se usara en determinadas condiciones y que contuviera por ejemplo una variable mal declarada.

Si al ejecutar el código no se llega a utilizar ese procedimiento, no se detectaría el error que contiene.

Para evitar “sorpresas” posteriores, es aconsejable realizar una pre-compilación del código.

Para realizarla podemos utilizar la opción de menú [**Compilar NombreDelFicheroAccess**] de la opción de menú [**Depuración**].

A esta opción de menú se puede llegar también mediante el botón [**Compilar**]  .

Esta pre-compilación revisa todo el código, e incluso posibles procedimientos que no serían utilizados durante la ejecución del programa. Esto nos da más garantía sobre la calidad del código y nos protege frente a ciertos tipos de error que de otra forma no podríamos detectar.

Errores en Tiempo de Ejecución

Hay una serie de errores que se pueden producir durante la ejecución del código, que no son de sintaxis ni originados por código incompleto o declaraciones inadecuadas.

Son, por ejemplo los errores provenientes de valores no previstos por el código pasados ya sea por el propio usuario extraídos de tablas, ficheros u otras fuentes de origen.

Son los típicos errores de Tiempo de Ejecución.

Supongamos que tenemos que dividir entre sí dos cantidades; si el denominador vale cero, nos dará un error de división entre cero.

Podría ocurrir que no hayamos previsto que un cuadro de texto contuviera el valor `Null`. Esto nos podría generar error al intentar asignar este valor a una cadena de texto.

También podría ocurrir que en una expresión por la que queremos asignar el resultado de una operación a una variable, ese resultado superara el rango admisible por el tipo de la variable, con lo que tendríamos un error de **Desbordamiento**.

Un programa profesional debe adelantarse a todas estas posibilidades.

Por ejemplo, si tenemos que dividir dos números, se debería comprobar que el denominador no contuviese el valor Cero.

Si tuviéramos que obtener, un elemento de una matriz, ó colección, podríamos evitar un error de subíndice fuera de intervalo.

Hay muchas circunstancias en las que se pueden producir errores en tiempo de ejecución.

VBA nos provee de una herramienta para poder controlarlos una vez que se han producido.

Supongamos que tenemos el siguiente procedimiento.

```
Public Sub ErrorDivisionPorCero()
   Dim n As Byte
   n = 0
   Debug.Print 4 / n
End Sub
```

- Sí, ya se que es un código sin mucho sentido; lo pongo sólo como ejemplo didáctico.

Si se ejecutara este procedimiento, ya sea llamado desde la ventana **Inmediato** o desde cualquier punto de la aplicación se producirá el preceptivo error:

Inmediatamente se interrumpirá la aplicación.

Si esto nos ocurriera a nosotros mientras estamos haciendo pruebas no tendría mayor importancia. Pero si le ocurriera a nuestro cliente, no creo que nos llamara para felicitarnos.

Por lo tanto deberemos anticiparnos a cualquier error que se pueda originar durante la ejecución de nuestros programas.

Tan importante como la redacción previa de los Casos de Uso que nos ayudarán a definir nuestro programa, es la redacción de las pruebas a las que deberemos someter nuestro programa antes de entregar al cliente la nueva versión. Y no sólo redactarlas, sino también ponerlas en práctica.

Frente a la posibilidad de un error, hay dos principales caminos

- Realizar todo tipo de filtros en el código para evitar que llegue a darse una situación de error
- En caso de que el error se produzca, capturarlo y darle una respuesta “civilizada”.

Y sobre todo acordarse de la ley fundamental de la programación:

- Si en un segmento de código existe la más pequeña posibilidad de que se produzca un error, éste acabará produciéndose irremediablemente, y además en el momento más inoportuno y de peores consecuencias.

Para capturar un error, VBA utiliza la más denostada, criticada y repudiada de las sentencias de código. Es la sentencia **`Goto`**.

Es la instrucción de salto que nos faltaba por ver en las dos entregas anteriores.

Instrucciones de salto.

La Instrucción Goto

Cuando el código se encuentra con esta sentencia realiza un salto incondicional.

Su sintaxis tiene esta estructura:

```
GoTo línea
```

Línea puede ser una Etiqueta de línea ó un número de línea.

Observa este código

```
Public Sub PruebaGoto()
    GoTo Etiqueta_01
    Debug.Print "No he saltado a Etiqueta_01"
Etiqueta_01:
    Debug.Print "*** Salto a la Etiqueta_01 ***"
    GoTo 10
    Debug.Print "No he saltado a 10"
10      Debug.Print "*** Salto a la línea 10 ***"
End Sub
```

Si ejecutamos el código nos imprime en la ventana Inmediato

```
*** Salto a la Etiqueta_01 ***
*** Salto a la línea 10 ***
```

Una etiqueta es una palabra que comience con una letra y termine con los dos puntos.

Un número de línea es un número situado al principio de la línea.

La utilización de números de línea proviene de la época de Basic, cuando las líneas de código debían ir precedidas de un número que las identificara.

```
GoTo Etiqueta_01
```

Salta a la etiqueta sin ejecutar la línea intermedia.

```
GoTo 10
```

Salta a la línea precedida por el número 10.

Gosub - - Return

Es otra instrucción de salto, de la cual sólo voy a comentar que existe por compatibilidad con las antiguas versiones de Basic.

Si alguien quiere más información puede acudir a la ayuda de VBA.

Personalmente desaconsejo completamente su uso, ya que en VBA existen alternativas más eficientes y claras.

La utilización de **Goto** y **Gosub** se desaconseja ya que pueden convertir el código en una sucesión de saltos de canguro imposible de seguir de una forma coherente.

Capturar Errores

Ya hemos comentado que durante la ejecución de una aplicación pueden producirse diversos tipos de errores, como rangos de valores no válidos, división por cero, manipulación de un elemento de una matriz, colección, o un fichero que no existan, etc.

Si prevemos que en un procedimiento pudiera producirse un error, para poder gestionarlo, pondremos en su cabecera o en un punto anterior al lugar donde el error se pudiera generar, la sentencia:

```
On error Goto Etiqueta
```

Si un procedimiento incluye esta línea, y en tiempo de ejecución, en una línea posterior a ésta, se produjera un error, la ejecución del código saltará a la línea que incluye esa etiqueta, y mediante el objeto **Err** podremos controlar el error.

El objeto Err

Este objeto contiene la información de los errores que se producen en tiempo de ejecución.

Cuando se produce un error, el objeto o el procedimiento que lo ha generado puede asignar datos a sus propiedades.

Las propiedades más importantes, o las que en este nivel nos interesan más, son:

- Number
- Description
- Source

Hay otras propiedades que, de momento no vamos a analizar, como son

HelpContext HelpFile LastDLLError

La propiedad **Number** contiene un número que sirve como identificador del error.

Description incluye una cadena de texto que nos sirve para interpretar las características del error.

Source contiene información sobre el proyecto u objeto que ha generado el error.

Tomando como modelo el código que generaba un error de División por cero, vamos a hacer otro procedimiento que lo controle:

```
Public Sub ErrorControlado()
   On Error GoTo HayError
   Dim n As Byte
   n = 0
   Debug.Print 4 / n
Salir:
    Exit Sub
HayError:
   Debug.Print "Error n° " & Err.Number
   Debug.Print Err.Description
   Debug.Print Err.Source
   Resume Salir
End Sub
```

El resultado de la ejecución de este código es:

```
Error nº 11
División por cero
Entrega11
```

Para usar las propiedades del objeto **Err**, se utiliza la misma sintaxis que para otros objetos `NombreObjeto.Propiedad`

La instrucción Resume

Hace que la ejecución del código continúe en una línea determinada, tras gestionar un error.

En el ejemplo anterior Resume Salir hace que el código vaya a la línea marcada con la etiqueta **Salir:**

La ejecución de

```
Resume Salir
```

hará que se salga del procedimiento.

Después de la palabra Resume, puede ponerse una etiqueta ó un número de línea.

En ambos casos se producirá una salto hasta la línea especificada.

Si se pusiera

```
Resume          ó          Resume 0
```

Si el error se hubiera producido en el procedimiento que contiene el controlador de errores, la ejecución continúa en la instrucción que lo causó.

Si el error se produjera en un procedimiento llamado, la ejecución continuará en la instrucción para el control de errores desde la cual se llamó al procedimiento que contiene la rutina de gestión de errores..

Si se hubiera escrito

```
Resume Next
```

Se ejecutará la línea siguiente a aquélla en la que se produjo el error ó se llamó al procedimiento para la gestión de errores.

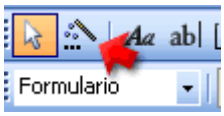
Resume sólo se puede usar en una rutina de gestión de errores.

Vamos a ver cómo manejan las excepciones los asistentes de Access.

Creamos un nuevo formulario mediante [**Nuevo**] - [**Vista Diseño**]. Con ello se abre un nuevo formulario en Vista Diseño, sin estar ligado a ningún origen de datos.

Si no tuviéramos visible la barra de herramientas, vamos a activarla, mediante la opción de menú [**Ver**] – [**Cuadro de herramientas**].

En esta barra, si no estuviese activado, activaremos el botón [Asistentes para controles]. En este gráfico ejemplo el botón está desactivado →



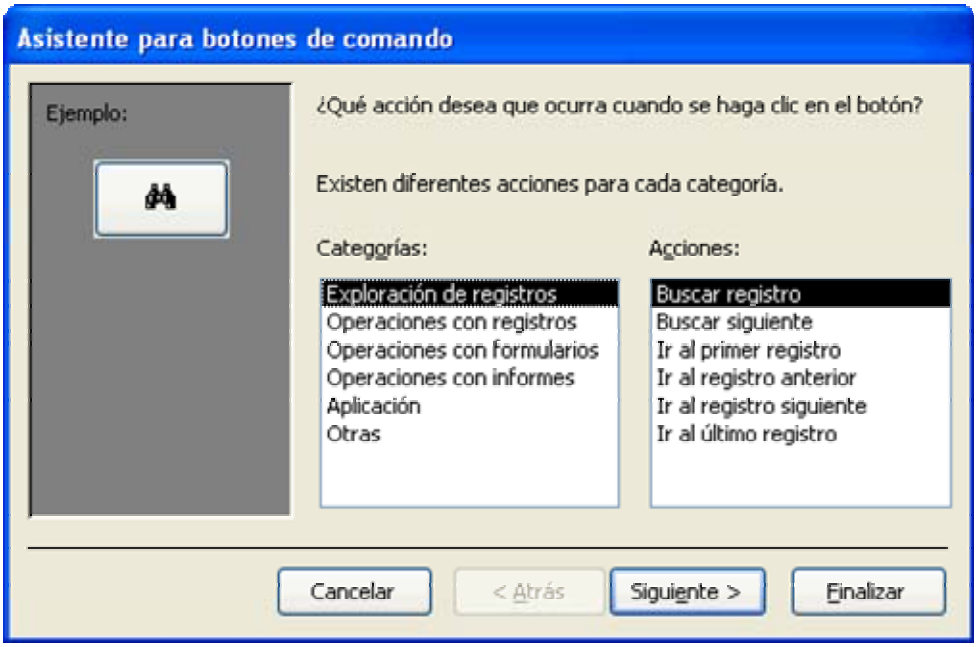
Una vez activado el botón de los asistentes, al seleccionar un control, Access generará código por nosotros, código que se ejecutará con alguno de los eventos del control.

Ahora vamos a poner un botón que servirá para cerrar el formulario cuando se presione sobre él.

Seleccionamos en la barra de herramientas el [Botón de comando] y lo colocamos en el formulario, pinchando sobre él y marcando, sin soltar el botón izquierdo, el rectángulo que va a ocupar el botón.

Tras hacer esto, se nos abre la ventana del Asistente para [Botones de comando].

En la parte de la izquierda nos muestra un ejemplo de los iconos que se podrían poner en el botón.



A continuación hay una lista de categorías de acciones.

La lista de la derecha muestra las acciones concretas que tiene el asistente para cada categoría.

Como lo que queremos es que el botón **Cierre el Formulario**, en la lista de la izquierda seleccionaremos Operaciones con formularios.

Automáticamente nos mostrará las acciones disponibles con los formularios.

Seleccionaremos en la lista de la derecha la opción **Cerrar formulario**.

Pulsamos en el botón [**Siguiente**].

Ahora nos aparecen nuevas opciones. En concreto nos permite seleccionar si queremos poner un texto ó una imagen en el botón.

Si seleccionáramos [**Texto**] podríamos cambiar el texto que nos sugiere Access.

Esta vez vamos a seleccionar una Imagen. Al seleccionar el botón [**Imagen**] se activan el botón [**Examinar**] y la Casilla de verificación [**Mostrar todas las imágenes**].

El botón [Examinar] permite seleccionar cualquier imagen compatible que tengamos en el ordenador. La casilla [Mostrar todas las imágenes] nos muestra los nombres de todas las imágenes prediseñadas por Access.

No vamos a hacer caso de estas opciones y seleccionaremos sin más la imagen Salir, que nos muestra una Puerta entreabierta señalada con una flecha.

Presionamos el botón [Siguiente] y nos pedirá un nombre para ese botón.

Le vamos a poner como nombre **cmdsalir**.

Veamos qué nos ha hecho el asistente.

Seleccionamos el botón, [Código] ó la opción de menó [Ver] - [Código].

Vemos que nos ha colocado, en el módulo de clase del formulario, el procedimiento

```
Private Sub cmdSalir_Click()
```

Este procedimiento es el Manejador del evento **Click** para el botón **cmdSalir**.

```
Private Sub cmdSalir_Click()
On Error GoTo Err_cmdSalir_Click

    DoCmd.Close

Exit_cmdSalir_Click:
    Exit Sub

Err_cmdSalir_Click:
    MsgBox Err.Description
    Resume Exit_cmdSalir_Click

End Sub
```

Lo primero que hace es utilizar `On Error GoTo Err_cmdSalir_Click` ; con ello si se produjera algún error saltará a la línea marcada con `Err_cmdSalir_Click:`

Esta es la forma como los asistentes de Access suelen nombrar las etiquetas que señalan el comienzo del código de Gestión de Errores **Err_NombreDelProcedimiento:**

A continuación nos encontramos con que se ejecuta el método **Close** del objeto **DoCmd**.

El objeto **DoCmd** es un objeto especial de Access. Contiene un gran número de métodos para ejecutar muchas de las tareas habituales con los objetos de Access.

A gran parte de estos métodos se les puede pasar una serie de argumentos.

Por sus amplias posibilidades, **DoCmd** merece un capítulo aparte. Lo desarrollaremos en próximas entregas.

Basta decir que en este caso, al llamar al método **Close** sin pasarle ningún argumento, **DoCmd** cierra la **ventana activa** y como ésta es el formulario del botón, al pulsarlo cierra el formulario, que es lo que queríamos conseguir.

Si no ha habido ningún problema, la ejecución del código se encuentra con la etiqueta que marca el punto a partir del cual se sale del procedimiento. Como aclaración, una etiqueta no ejecuta ninguna acción, sólo indica dónde comienza un segmento de código. En nuestro caso, a partir de este punto nos encontramos con la línea `Exit Sub` que nos hace salir del procedimiento `cmdSalir_Click`.

Después de esta última línea comienza el segmento de código que controla cualquier error en tiempo de ejecución que se pudiera originar dentro del procedimiento.

Primero, mediante un cuadro de mensaje, nos muestra el texto contenido en la propiedad **Descripción** del objeto **Err**.

A continuación, mediante `Resume Exit_cmdSalir_Click` hace que salte el código a la etiqueta que marca la salida del procedimiento.

Una puntualización: `Resume`, además de dar la orden de que el código se siga ejecutando desde un determinado punto del procedimiento, pone a “cero” las propiedades del objeto **Err**, lo que equivale a hacer que desaparezca el error.

Como hemos visto, cuando usamos un asistente para controles de Access, en el código generado se suele colocar un segmento para la gestión de errores.

Este código escrito por Access lo podemos cambiar; por ejemplo podríamos sustituir `Exit_cmdSalir_Click` por **`Salir`**, ó `Err_NombreDelProcedimiento` por **`hayError`**, e incluso lo podríamos quitar, si estuviéramos seguros de que nunca se podría llegar a producir un error o ya tuviéramos desarrollado nuestro propio sistema para la gestión de errores.

Gestionando errores

Supongamos que nos han encargado un programa para visualizar en un formulario el contenido, en formato texto, de los ficheros que seleccionemos.

Para especificar el fichero que se va a visualizar nos piden que su nombre, incluida su ruta, se escriba en un cuadro de texto.

Tras esto, y presionar un botón, su contenido se mostrará en un segundo cuadro de texto.

Se decide seleccionar un cuadro de texto como soporte, con el fin de que se puedan seleccionar y copiar segmentos del texto del contenido de los ficheros.

> *Debo aclarar que no todo el contenido de un fichero cualquiera se podrá mostrar en un cuadro de texto. Si el fichero es de tipo Binario, gráficos, ficheros exe ó multimedia, habrá caracteres* que no se muestren ó que tengan secuencias ininteligibles.

Como ya habíamos escrito previamente un procedimiento para mostrar el contenido de ficheros (**`MuestraFichero`** del capítulo anterior) vamos a aprovecharlo, con algunas modificaciones para adaptarlo a los nuevos requerimientos.

El nuevo procedimiento exigirá que se le pase como parámetro un cuadro de texto; en este caso con nombre **`Pizarra`**, que será donde se irá escribiendo el contenido del fichero mientras se vaya leyendo.

Además le añadiremos un control de errores, ya que podría ocurrir que no se pudiera abrir el fichero por no existir, o porque estuviera abierto en modo exclusivo por algún usuario.

```
Public Sub MuestraFichero( _
            ByVal Fichero As String, _
            ByRef Pizarra As TextBox)
    On Error GoTo ProducidoError
        Dim intFichero As Integer
    Dim strLinea As String
    intFichero = FreeFile
    Open Fichero For Input As #intFichero
    While Not EOF(intFichero)
        Line Input #intFichero, strLinea
        Pizarra.Value = Pizarra.Value & strLinea
    Wend
Salir:
    Exit Sub
ProducidoError:
    MsgBox "Se ha producido el error " & Err.Number, _
            vbCritical + vbOKOnly, _
            "Error en el procedimiento MuestraFichero()"
```

```
    Pizarra.Value = ""
    Resume Salir
    End Sub
```

Creamos un nuevo formulario y le añadimos los siguientes elementos

Control	Nombre
Cuadro de texto	txtFichero
Cuadro de texto	txtContenido
Botón	cmdVerFichero

El cuadro de texto **txtFichero** servirá para introducir el nombre completo del fichero a visualizar.

En **txtContenido** mostraremos el contenido del fichero.

El botón **cmdVerFichero** activará el procedimiento que sirve para mostrar el contenido del fichero. La llamada al procedimiento lo colocaremos en el evento Al hacer clic del botón **cmdVerFichero**.

Usaremos el procedimiento `MuestraFichero` al que le pasaremos como parámetro el control **txtContenido**.

En el cuadro de texto **txtContenido** activamos la Barra Vertical en la propiedad **Barras de Desplazamiento**.

En el evento Al hacer clic del botón escribiremos lo siguiente:

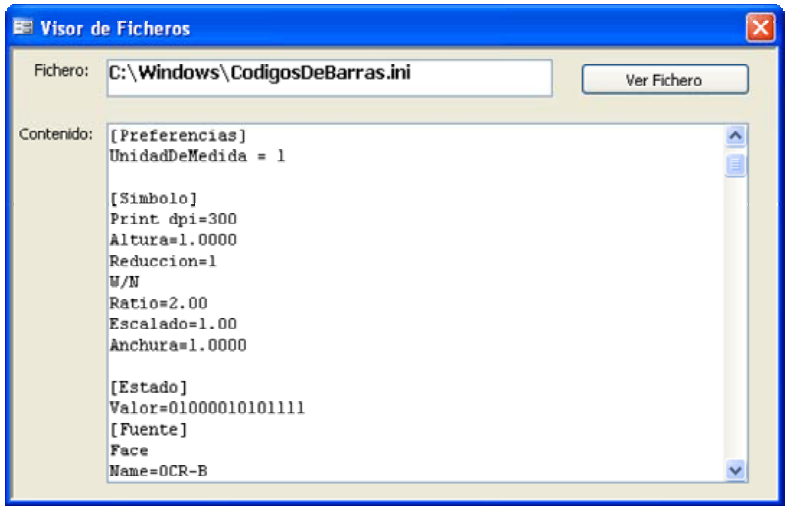
```
Private Sub cmdVerFichero_Click()
    txtContenido = ""
    MuestraFichero txtFichero, txtContenido
End Sub
```

Lo que hace el procedimiento es:

Limpia el posible contenido de `txtContenido` asignándole una cadena vacía.

Llama al procedimiento `MuestraFichero` Pasándole como parámetros el nombre del fichero que queremos abrir, contenido en el control `txtFichero`, y el control en el que queremos mostrar los datos `txtContenido`.

En el ejemplo se muestra el contenido del fichero CodigosDeBarras.ini ubicado en la carpeta Windows.



Generación directa de errores (Err.Raise)

Con el método **Raise** del objeto **Err**, podemos forzar la generación de un error en tiempo de ejecución

La sintaxis del método **Raise** es

Err.**Raise** ***NúmeroError***, ***OrigenError***, ***Descripción***, **FicheroAyuda**, ***ContextoDeAyuda***

NúmeroError es el número de error que queremos pasar al objeto Err.

OrigenError Objeto, Aplicación, Procedimiento que genera el error.

Descripción Texto que identifica al propio error.

FicheroAyuda Fichero de ayuda con información sobre el error.

ContextoAyuda Identificador de contexto que especifica el tema del archivo indicado en FicheroAyuda, que contiene la información de ayuda del error.

De los cinco parámetros, el único obligatorio es el Número que identifica al error.

A título de información, adelanto que con Access se pueden utilizar ficheros de ayuda creados al efecto por nosotros.

La creación y utilización de ficheros de ayuda es uno de los temas considerados como "avanzados".

Los números de error reservados para ser definidos por el usuario son los que están entre el número **513** y el **65535**.

```
Public Sub GenerarError()
       On Error GoTo HayError
   ' Aquí generamos el error
   Err.Raise 666, "GenerarError", _
              "¡Houston! Tenemos un problema"
Salir:
    Exit Sub
HayError:
   Debug.Print "Error nº " & Err.Number
   Debug.Print "Se ha producido el error: " _
            & Err.Description
   Debug.Print "en el procedimiento: " _
             ; Err.Source
   Resume Salir
End Sub
```

Si ejecutamos el procedimiento **GenerarError** nos mostrará en la ventana Inmediato:

```
Error nº 666
Se ha producido el error: ¡Houston! Tenemos un problema
en el procedimiento: GenerarError
```