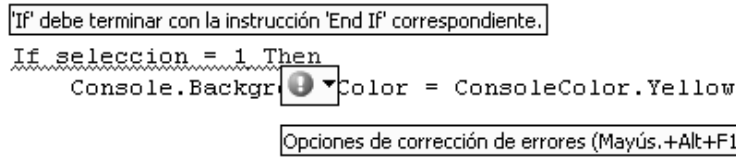
A. Los diferentes tipos de errores

Para un desarrollador, los errores son una de las principales fuentes de estrés. En realidad, podemos clasificar estos errores en tres categorías. Vamos a ver cada uno de ellos así como las soluciones disponibles para resolverlos.

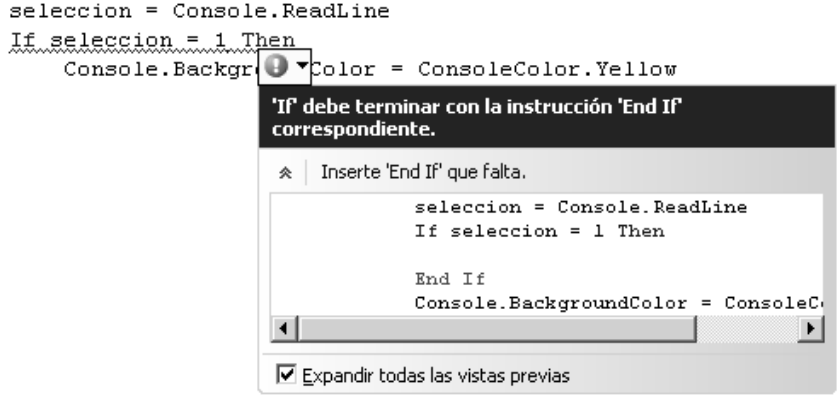
1. Los errores de sintaxis

Este tipo de errores se produce en el momento de la compilación, cuando una palabra clave del lenguaje está mal escrita. Eran muy frecuentes con las primeras herramientas de desarrollo, en las que el editor de código y el compilador eran dos entidades separadas. Ahora son cada vez más raros en los entornos similares a Visual Studio. La mayoría de estos entornos ofrecen un análisis sintáctico mientras se escribe el código. Desde este punto de vista, Visual Studio proporciona muchas funcionalidades que nos permiten eliminar este tipo de errores.

Así, por ejemplo, controla que cada instrucción `If` se acabe correctamente con un `End If`.

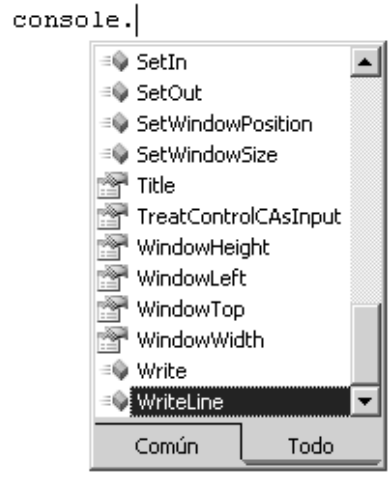


Si se detecta un error de sintaxis, Visual Basic propone las posibles soluciones para corregirlo. Se pueden ver haciendo clic en el icono asociado al error.

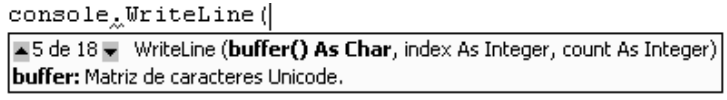


Por otra parte, los "errores de ortografía" en los nombres de propiedades o métodos se eliminan fácilmente gracias a las funcionalidades `IntelliSense`. `IntelliSense` se encarga de:

- Mostrar automáticamente la lista de los miembros disponibles.



- Mostrar la lista de parámetros a facilitar para la llamada de un procedimiento o función:



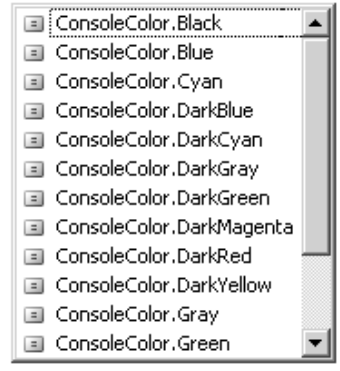
Si un método está sobrecargado, `IntelliSense` muestra su número y le permite recorrerlas utilizando las flechas arriba y abajo del teclado.

- Mostrar información puntual acerca de los miembros de una clase:

- Completar automáticamente las palabras: empiece por escribir el principio de palabra, luego utilice la combinación de teclas [Ctrl][Espacio] para mostrar todo lo que puede utilizar como palabra en ese emplazamiento, que empieza con los caracteres ya elegidos. Si sólo hay una posibilidad, la palabra se añade automáticamente; si no, selecciónela de la lista y valide con la tecla [Tab].

- Mostrar la lista de los posibles valores para una propiedad de tipo enumeración.

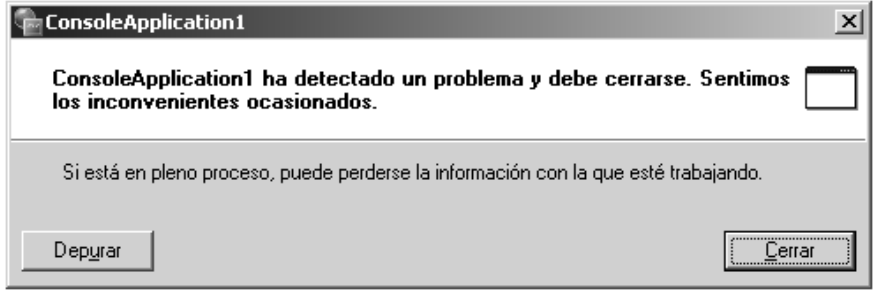
```
If seleccion = 1 Then
    Console.BackgroundColor = |

End If
```



Con todas estas funcionalidades, es prácticamente imposible generar errores de sintaxis en el código.

2. Los errores de ejecución

Estos errores aparecen después de la compilación cuando decide lanzar la ejecución de su aplicación. La sintaxis del código es correcta pero el entorno de su aplicación no permite la ejecución de una instrucción utilizada en su aplicación. Esto ocurre, por ejemplo, cuando se intenta abrir un archivo que no existe en el disco de su máquina. Seguramente obtenga un cuadro de diálogo de este tipo, no muy agradable para el usuario:



Afortunalmente, Visual Basic permite la recuperación de este tipo de error y evita así la visualización de este inquietante cuadro de diálogo. Existen dos técnicas para la gestión de este tipo de errores:

- la gestión en línea;
- las excepciones.

Las veremos en detalle más adelante en este capítulo.

Los errores de lógica

Los peores enemigos de los desarrolladores. Todo se compila sin problema, todo se ejecuta sin problema y sin embargo "¡no funciona!!"

En este caso conviene revisar la lógica de funcionamiento de la aplicación. Las herramientas de depuración nos permiten seguir el desarrollo de la aplicación, ubicar puntos de parada, visualizar el contenido de las variables, etc.

B. Tratamiento de los errores

Existen dos técnicas para el tratamiento de los errores de Visual Basic:

- la gestión en línea;
- el tratamiento de la excepción.

1. La gestión en línea

La instrucción `On Error` es el elemento básico de la gestión de los errores en línea. Cuando esta instrucción se ejecuta en un procedimiento o función, activa la gestión de los errores para ese procedimiento o función. Sin embargo, conviene indicar a nuestro gestor cómo debe reaccionar cuando una instrucción desencadena un error. Dos soluciones:

```
On error resume next
```

La ejecución del código seguirá en la línea posterior a la que ha provocado el error.

```
On error goto etiqueta
```

La ejecución del código pasará a la línea señalada con `etiqueta`.

La sintaxis es por tanto la siguiente:

```
Private Sub Nombre_de_procedimiento()
   On Error Goto gestionErrores
      ...
      "Instrucciones peligrosas"
  ...
Exit Sub
gestionErrores:
  ...
  código que gestiona los errores
  ...
End Sub
```

La etiqueta hacia la que el gestor de errores dirigirá la ejecución se debe encontrar en el mismo procedimiento que la instrucción `on error goto`. La instrucción `Exit sub` es obligatoria para que el código de gestión de errores no se ejecute después de las instrucciones normales, sino sólo en el caso de errores.

El código del gestor de errores debe determinar el comportamiento a seguir en el caso de error.

Tres soluciones:

`Resume`

Se vuelve a intentar la ejecución de la línea que ha producido el error.

`resume next`

Se sigue con la ejecución en la línea que sigue a la que ha provocado el error.

`exit sub` o `exit function`

Se para la ejecución de este procedimiento o función.

Normalmente, el gestor de errores muestra un cuadro de diálogo que le pregunta al usuario qué quiere hacer.

En función de su respuesta, se utilizará una solución u otra.

```
Private Sub AbrirArchivo()
        On Error GoTo gestionErrores
        My.Computer.FileSystem.OpenTextFileReader("c:\prueba")
        Exit Sub
gestionErrores:
        Dim respuesta As Integer
        respuesta = MsgBox("no se puede leer el archivo",
MsgBoxStyle.AbortRetryIgnore)
        Select Case respuesta
            Case MsgBoxResult.Retry
                Resume
            Case MsgBoxResult.Ignore
                Resume Next
            Case MsgBoxResult.Abort
                Exit Sub
        End Select
    End Sub
```

Nos queda todavía un problema por resolver: nuestro gestor de errores reaccionará independientemente del error. Para poder determinar qué error se acaba de producir en la aplicación, tenemos a nuestra disposición el objeto `err` que nos proporciona información sobre el último error que ha aparecido. Este objeto contiene, entre otras, dos propiedades, `number` y `description`, que nos permiten obtener el código del error y su descripción. Podemos entonces modificar nuestro código para que reaccione de manera diferente en función del error.

```
Private Sub AbrirArchivoBis()
        On Error GoTo gestionErrores
        My.Computer.FileSystem.OpenTextFileReader("a:\prueba")
        Exit Sub
gestionErrores:
```

```
        Dim respuesta As Integer
        Console.WriteLine(Err.Number)
        Stop
        If Err.Number = 53 Then
            respuesta = MsgBox("no se ha encontrado el archivo", MsgBoxStyle.
AbortRetryIgnore)
            Select Case respuesta
                Case MsgBoxResult.Retry
                    Resume
                Case MsgBoxResult.Ignore
                    Resume Next
                Case MsgBoxResult.Abort
                    Exit Sub
            End Select
        End If
        If Err.Number = 57 Then
            respuesta = MsgBox("insertar un disco en el lector",
MsgBoxStyle.OKCancel)
            Select Case respuesta
                Case MsgBoxResult.OK
                    Resume
                Case MsgBoxResult.Cancel
                    Exit Sub
            End Select
        End If
    End Sub
```

Se puede desactivar un gestor de errores utilizando la instrucción `on error goto 0`. Si se produce un error después de esta instrucción, no se recupera y la aplicación se detiene.

De hecho, la aplicación no se para inmediatamente, sino que Visual Basic busca en las funciones de llamada si hay un gestor de errores activo. Si encuentra uno, le encarga la gestión del error.

```
Private Sub procedimiento1()
        On Error GoTo gestionErrores
        procedimiento2()
        Exit Sub
gestionErrores:
        MsgBox("error de ejecución")
    End Sub
    Private Sub procedimiento2()
        Dim x, y As Integer
        x = 0
        y = (1 / x)
    End Sub
```