xaScript Help file

http://www.ozs.com

Guía del Usuario

por OZ Software

xaConta

xaScript Help file

Impreso en Madrid en: enero 2019.

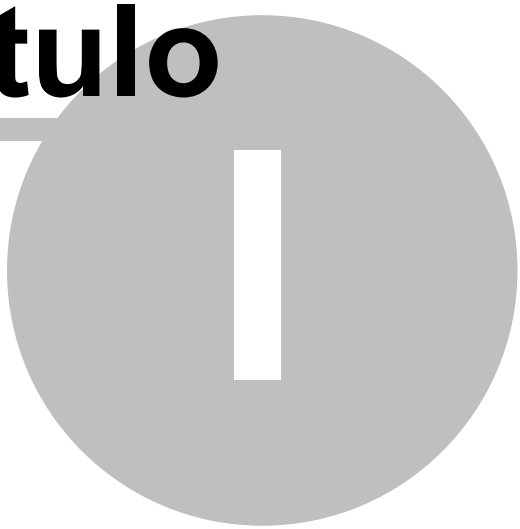
Agradecimientos:

A todos los usuarios que han participado en la fase Beta del producto aportandonos valiosa información sobre errores y sugirriendo mejoras que han podido incorporarse al producto.

Por último, a todos los usuarios de OZ Software que nos han apoyado y animado desde hace muchos años.

Tabla de Contenido

Capítulo



1 xaScript

Copyright:

xaScript es un producto de OZ Software (http://www.ozs.com)
Soporte en http://www.ozs.com/?q=blog

Introduction:

xaScript de OZ Software es una potente herramienta de scripting que abre completamente nuevas posibilidades a los desarrolladores de [x]Harbour. xaScript le permite crear código fuente [x]Harbour desde sus propios programas que luego podrá ser evaluado en tiempo de ejecución sin necesidad de utilizar para nada el propio compilador de [x]Harbour o un enlazador.

Esta nueva herramienta lleva las posibilidad de la clásica 'Macro' al máximo nivel. Con xaScript puede crear aplicaciones que cambien completamente su interface en tiempo de ejecución, o que se ajusten a las preferencias individuales de cada uno de sus clientes sin necesidad de tener una versión individual de su aplicación para cada uno de ellos.

xaScript es una pequeña librería que puede ser enlazada con su aplicación para ofrecerle toda la potencia del scripting.

Son tan sólo tres pasos los necesarios para ejecutar cualquier script, que son:

1. oScript := TScript():New(cTexto)
2. oScript:Compile()
3. oScript:Run()

¡¡ Eso es todo !!

Un archivo script puede contener más de una función, incluso puede añadir a un script todos los archivos de código fuente que desee. Absolutamente todas las funciones incluidas en todos los archivos, al igual que cualquier otra función existente en su aplicación, son accesibles desde el scripting.

xaScript detecta todos los errrores de sintaxis en el proceso de compilación, pero no se ejecuta ningún código durante ese proceso.

Antes de llamar al método Run() deberá comprobar de que no se ha producido ningún error de compilación. Para dicho motivo el objeto TScript posee la propiedad **cError** que guarda el último error encontrado, y también existe un método **MsgError** que le muestra en pantalla el error en el clásico diálogo MsgInfo.

Puede pasar hasta 10 parametros a cualquier función, por ejemplo:

```
xRet := oScript:Compile( "Test", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 )
```

Esta expresión ejecutará la función 'Test' con diez parametros numéricos y la variable xRet retendrá el valor retornado por la función.

Al igual que en el proceso de compilación, si durante la ejecución se produjese algún problema, la propiedad **oScript:cError** retornará información sobre el error.

Puede crear variables globales (Public) de la misma forma que lo hace en [x]Harbour pero con la instrucción STATIC al principio del módulo a compilar:

```
STATIC a := 1
FUNCTION ....
  ...
  ...
RETURN ...
```

o bien utilizando la clásica cláusula PUBLIC.

Este tipo de variables son accesibles desde cualquier función dentro del script. También puede crear variables de ámbito local con las instrucciones LOCAL o STATIC pero ya dentro de la declaración de la función. Las variables estáticas retienen su valor entre diferentes llamadas.

```
FUNCTION Test()
  Static cText := "hello", cFile := "reamdme.txt"
  Local xVar := 1
  ..........
  ..........
RETURN .T.
```

Puede incluir igualmente cualquier comentario en su script, de la misma manera que lo hace en [x]Harbour con "//".

Todos los operadores aritméticos y booleanos de [x]Harbour son soportados.

Las siguientes instrucciones son soportadas y pueden ser anidadas a cualquier nivel:

- IF, ELSEIF, ELSE, ENDIF
- DO CASE, CASE, OTHERWISE, ENDCASE
- DO WHILE, END WHILE
- FOR, NEXT
- LOOP, EXIT
- RETURN
- WITH Ob jeto, END WITH
- FOR EACH, NEXT
- TRY, CATCH, END

Puede ejecutar cualquier función de [x]Harbour desde su archivo Script siempre que dicha función este enlazada con su aplicación. Si usted recibe el mensaje de error: 'Function not found', lo único que tiene que hacer es utilizar la instrucción REQUEST para forzar que la función se adjunte al ejecutable. Por ejemplo:

```
REQUEST Descend, Alltrim
```

Esto hará acccesible las funciones Descend y Alltrim en su archivo script incluso aunque no se usen en ninguna parte de su aplicación.

Para acceder a cualquier variable de su aplicación deberá crear una variable pública que guarde su valor.

System requirements:

xaScript funciona con Harbour y [x]Harbour. A partir de la versión 1.7 sólo se entregan las librerías para Harbour con BCC y MinGW. No obstante puede reconstruir las librerías con [x]Harbour y cualquier compilador sin problemas ya que se entregan todos los fuentes del producto.

El software de acompañamiento de xaScript esta realizado con Xailer, sin embargo no es necesario en absoluto Xailer para utilizar xaScript

Known limitations:

La funcionalidad de xaScript está basada en el soporte de 'marcos' que incluye el propio [x]Harbour, por lo que hereda todos sus ventajas y limitaciones. xaScript no es un sustituto de [x]Harbour ni debe userse de esa forma, tan sólo lleva el soporte de macros de [x]Harbour a su máximo nivel.

El ánimo de xaScript es soportar en su totalidad todas las características del lenguaje CA-Clipper. Algunas de las extensiones del lenguaje introducidas por [x]Harbour son soportadas, pero no todas en su totalidad.

Las actuales limitaciones conocidas de xaScript son:

- Recursividad en funciones de scripting no está testado y no se recomienda su uso
- Directiva BEGIN SEQUENCE no soportada
- Cláusula FINALLY en bloques TRY-CATCH no soportado
- Comando SWITCH no soportado, deberá usar DO CASE en su vez

Linking:

Simplemente incluya la librería en su fichero de enlazado. Si utiliza el compilador de MingGw enlace la librería libxaScript.a, si utiliza Borland utilice xaScript.lib

A partir de la versión 1.7 sólo se entregan las librerías para Harbour con BCC y MinGW. No obstante puede reconstruir las librerías con [x]Harbour y cualquier compilador sin problemas ya que se entregan todos los fuentes del producto.

¡¡Eso es todoll!

Classes:

Toda la potencia de xaScript se encuentra encapsulada en tres clases:

- TScript
- TScrFun
- TScrLin

TScript:

TScript es la clase principal de xaScript y en condiciones normales será la única que clases que realmente usara. TScript es un contenedor de controls que engloba todas las funciones de scripting y las variables públicas creadas por usted.

Usted puede crear múltipiles instancias de TScript y compilar tantos módulos como desee con ellas. Todas las funciones y variables serán accesibles por cualquier función de scripting. Si intentase incluir una función ya existente simprlemente será reemplazada.

Los métodos más importantes de esta clase son su contructor New() y el método Compile(). El primero crea una instancia de TScript y permite pasar como parametro el texto a compilar. El segundo método realiza el proceso de compilado e igualmente puede pasar como parametro un texto diferente a compilar. Puede ejecutar repetidamente el método Compile() incluyendo todo el código fuente que desee.

La propiedad más importante de TScript es cError que guarda la descripción del último error producido.

TScrFun:

Esta clase encapsula cada función existente en el script. Usted no requerirá normalmente acceder a esta clase ya que puede acceder a ella directamente desde la clase TScript.

TScrLin:

Esta clase encapsula cada línea compilada de un objeto TScrFun. Normalmente no necesitará acceder par nada a esta clase.

1.1 Copyright

xaScript es un producto de OZ Software (http://www.ozs.com)
Soporte en http://www.ozs.com/?q=blog

1.2 Introducción

xaScript de OZ Software es una potente herramienta de scripting que abre completamente nuevas posibilidades a los desarrolladores de [x]Harbour. xaScript le permite crear código fuente [x]Harbour desde sus propios programas que luego podrá ser evaluado en tiempo de ejecución sin necesidad de utilizar para nada el propio compilador de [x]Harbour o un enlazador.

Esta nueva herramienta lleva las posibilidad de la clásica 'Macro' al máximo nivel. Con xaScript puede crear aplicaciones que cambien completamente su interface en tiempo de ejecución, o que se ajusten a las preferencias individuales de cada uno de sus clientes sin necesidad de tener una versión individual de su aplicación para cada uno de ellos.

xaScript es una pequeña librería que puede ser enlazada con su aplicación para ofrecerle toda la potencia del scripting.

Son tan sólo tres pasos los necesarios para ejecutar cualquier script, que son:

1. oScript := TScript():New(cTexto)
2. oScript:Compile()
3. oScript:Run()

¡¡ Eso es todo ll!

Un archivo script puede contener más de una función, incluso puede añadir a un script todos los archivos de código fuente que desee. Absolutamente todas las funciones incluidas en todos los archivos, al igual que cualquier otra función existente en su aplicación, son accesibles desde el scripting.

xaScript detecta todos los errrores de sintaxis en el proceso de compilación, pero no se ejecuta ningún código durante ese proceso.

Antes de llamar al método Run() deberá comprobar de que no se ha producido ningún error de compilación. Para dicho motivo el objeto TScript posee la propiedad **cError** que guarda el último error encontrado, y también existe un método **MsgError** que le muestra en pantalla el error en el clásico diálogo MsgInfo.

Puede pasar hasta 10 parametros a cualquier función, por ejemplo:

```
xRet := oScript:Compile( "Test", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 )
```

Esta expresión ejecutará la función 'Test' con diez parametros numéricos y la variable xRet retendrá el valor retornado por la función.

Al igual que en el proceso de compilación, si durante la ejecución se produjese algún problema, la propiedad **oScript:cError** retornará información sobre el error.

Puede crear variables globales (Public) de la misma forma que lo hace en [x]Harbour pero con la instrucción STATIC al principio del módulo a compilar:

```
STATIC a := 1
FUNCTION ....
  ...
  ...
RETURN ...
```

o bien utilizando la clásica cláusula PUBLIC.

Este tipo de variables son accesibles desde cualquier función dentro del script. También puede crear variables de ámbito local con las instrucciones LOCAL o STATIC pero ya dentro de la declaración de la función. Las variables estáticas retienen su valor entre diferentes llamadas.

```
FUNCTION Test()
  Static cText := "hello", cFile := "reamdme.txt"
  Local xVar := 1
  ..........
  ..........
RETURN .T.
```

Puede incluir igualmente cualquier comentario en su script, de la misma manera que lo hace en [x]Harbour con "//".

Todos los operadores aritméticos y booleanos de [x]Harbour son soportados.

Las siguientes instrucciones son soportadas y pueden ser anidadas a cualquier nivel:

- IF, ELSEIF, ELSE, ENDIF
- DO CASE, CASE, OTHERWISE, ENDCASE
- DO WHILE, END WHILE
- FOR, NEXT
- LOOP, EXIT
- RETURN
- WITH Ob jeto, END WITH
- FOR EACH, NEXT
- TRY, CATCH, END

Puede ejecutar cualquier función de [x]Harbour desde su archivo Script siempre que dicha función este enlazada con su aplicación. Si usted recibe el mensaje de error: 'Function not found', lo único que tiene que hacer es utilizar la instrucción REQUEST para forzar que la función se adjunte al ejecutable. Por ejemplo:

```
REQUEST Descend, Alltrim
```

Esto hará acccesible las funciones Descend y Alltrim en su archivo script incluso aunque no se usen en ninguna parte de su aplicación.

Para acceder a cualquier variable de su aplicación deberá crear una variable pública que guarde su valor.

1.3 Requerimientos

xaScript funciona con Harbour y [x]Harbour. A partir de la versión 1.7 sólo se entregan las librerías para Harbour con BCC y MinGW. No obstante puede reconstruir las librerías con [x]Harbour y cualquier compilador sin problemas ya que se entregan todos los fuentes del producto.

El software de acompañamiento de xaScript esta realizado con Xailer, sin embargo no es necesario en absoluto Xailer para utilizar xaScript

1.4 Limitaciones

La funcionalidad de xaScript está basada en el soporte de 'marcos' que incluye el propio [x]Harbour, por lo que hereda todos sus ventajas y limitaciones. xaScript no es un sustituto de [x]Harbour ni debe userse de esa forma, tan sólo lleva el soporte de macros de [x]Harbour a su máximo nivel.

El ánimo de xaScript es soportar en su totalidad todas las características del lenguaje CA-Clipper. Algunas de las extensiones del lenguaje introducidas por [x]Harbour son soportadas, pero no todas en su totalidad.

Las actuales limitaciones conocidas de xaScript son:

- Recursividad en funciones de scripting no está testado y no se recomienda su uso
- Directiva BEGIN SEQUENCE no soportada
- Cláusula FINALLY en bloques TRY-CATCH no soportado
- Comando SWITCH no soportado, deberá usar DO CASE en su vez

1.5 Enlazado

Simplemente incluya la librería en su fichero de enlazado. Si utiliza el compilador de MingGw enlace la librería libxaScript.a, si utiliza Borland utilice xaScript.lib

A partir de la versión 1.7 sólo se entregan las librerías para Harbour con BCC y MinGW. No obstante puede reconstruir las librerías con [x]Harbour y cualquier compilador sin problemas ya que se entregan todos los fuentes del producto.

¡¡Eso es todoll!

1.6 Actualizaciones

Versión 1.7: (Septiembre 2012)

- Soporte de Harbour 3.0
- Soporte de creación de clases con herencia usando las librerías de Harbour (permite ejecutar formularios creados por Xailer)

Versión 1.6: (Marzo 2012)

- Nueva propiedad TScript:lCheckFunctionExists
- Error en descripción correcta del error cuando éste se producía en posteriores funciones llamadas desde el script principal

1.7 Clases

Toda la potencia de xaScript se encuentra encapsulada en tres clases:

- TScript
- TScrFun
- TScrLin

TScript:
TScript es la clase principal de xaScript y en condiciones normales será la única que clases que realmente usara. TScript es un contenedor de controls que engloba todas las funciones de scripting y las variables públicas creadas por usted.

Usted puede crear múltipiles instancias de TScript y compilar tantos módulos como desee con ellas. Todas las funciones y variables serán accesibles por cualquier función de scripting. Si intentase incluir una función ya existente simprlemente será reemplazada.

Los métodos más importantes de esta clase son su contructor New() y el método Compile(). El primero crea una instancia de TScript y permite pasar como parametro el texto a compilar. El segundo método realiza el proceso de compilado e igualmente puede pasar como parametro un texto diferente a compilar. Puede ejecutar repetidamente el método Compile() incluyendo todo el código fuente que desee.

La propiedad más importante de TScript es cError que guarda la descripción del último error producido.

TScrFun:
Esta clase encapsula cada función existente en el script. Usted no requerirá normalmente acceder a esta clase ya que puede acceder a ella directamente desde la clase TScript.

TScrLin:
Esta clase encapsula cada línea compilada de un objeto TScrFun. Normalmente no necesitará acceder par nada a esta clase.

1.7.1 TScript

TScript es la clase principal de xaScript y en condiciones normales será la única que clases que realmente usara. TScript es un contenedor de controls que engloba todas las funciones de scripting y las variables públicas creadas por usted.

Usted puede crear múltipiles instancias de TScript y compilar tantos módulos como desee con ellas. Todas las funciones y variables serán accesibles por cualquier función de scripting. Si intentase incluir una función ya existente simprlemente será reemplazada.

Los métodos más importantes de esta clase son su contructor New() y el método Compile(). El primero crea una instancia de TScript y permite pasar como parametro el texto a compilar. El segundo método realiza el proceso de compilado e igualmente puede pasar como parametro un texto diferente a compilar. Puede ejecutar repetidamente el método Compile() incluyendo todo el código fuente que desee.

La propiedad más importante de TScript es cError que guarda la descripción del último error producido.

1.7.1.1 TScript:Data

■ Sólo lectura ■ Asignable

Ámbito	Nombre	Tipo	Valor inicial
■	cError	Carácter	""
■	cIncludePath	Carácter	""
■	cText	Carácter	""
■	lCheckFunctionExists	Lógico	.T.
■	lPreProcess	Lógico	.F.
■	lError	Lógico	.F.
■	oError	Objeto	NIL

1.7.1.1.1 TScript:cError

Guarda el último error producido. Deberá comprobar el valor de esta propiedad después de cualquier proceso de compilación.

Ámbito:	Sólo lectura
Tipo:	Carácter
Valor inicial:	""

1.7.1.1.2 TScript:cIncludePath

Directorios de búsqueda para ficheros include en el Preprocesador. Cada directorio debe separarse por un ';'.

Ámbito:	Asignable
Tipo:	Carácter
Valor inicial:	""

1.7.1.1.3 TScript:cText

Texto a compilar.

Ámbito:	Asignable
Tipo:	Carácter
Valor inicial:	""

1.7.1.1.4 TScript:lCheckFunctionExists

Si verdadero comprueba que las funciones que se incluyan en el Script no existen de antemano como funciones internas de la aplicación.

Ámbito:	Asignable
Tipo:	Lógico
Valor inicial:	.T.

1.7.1.1.5 TScript:lPreProcess

Si verdadero se preprocesará el código fuente. Es necesario que esta propiedad esté a verdadero si se utilizan las instrucciones #command, #define o se incluye cualquier archivo CH.

Ámbito:	Asignable
Tipo:	Lógico
Valor inicial:	.F.

1.7.1.1.6 TScript:lError

Verdadero si se ha producido algún error.

Ámbito:	Sólo lectura
Tipo:	Lógico
Valor inicial:	.F.

1.7.1.1.7 TScript:oError

Guarda el último objeto estándar Error de [x]Harbour generado por TScript.

Ámbito:	Sólo lectura
Tipo:	Objeto
Valor inicial:	NIL

1.7.1.2 TScript:Métodos

■ Constructor ■ Estándar

Tipo	Nombre
■	Compile
■	DelFunction
■	GetError
■	GetFunction
■	IsFunction
■	LoadPCode
■	MsgError
■	New
■	Reset
■	Run

- SavePCode

1.7.1.2.1 TScript:Compile

Este método compila el script. Si cText ha sido pasado como parametro, éste será compilado en vez del definido en el constructor New().

Si la propiedad lPreProcess está a verdadero el texto será preprocesado.

Tipo	Estándar
Parámetros	[<**cText**>]: Texto a compilar. Por defecto cText
Valor de retorno	<**lSuccess**>: Verdadero si éxito

1.7.1.2.2 TScript:DelFunction

Borrra la función cFunctionName de TScript.

Tipo	Estándar
Parámetros	<**cFunctionName**>: Función a borrar
Valor de retorno	<**lSuccess**>: Verdadero si éxito

1.7.1.2.3 TScript:GetError

Retorna una cadena con el mismo literal mostrado por el mensaje MsgError.

Tipo	Estándar
Parámetros	Ninguno
Valor de retorno	<**cError**> Mensaje de error

1.7.1.2.4 TScript:GetFunction

Retorna si existe el objeto TScrFun de cFunctionName, en caso contrario retorna NIL.

Tipo	Estándar
Parámetros	<**cFunctionName**>: Función a buscar
Valor de	<**oFunction**>

retorno	Objeto TScrFun

1.7.1.2.5 TScript:IsFunction

Retorna verdadero si la función cFunctionName existe en el Script.

Tipo	Estándar
Parámetros	<**cFunctionName**>: Función a buscar
Valor de retorno	<**lExists**>: Verdadero si existe

1.7.1.2.6 TScript:LoadPCode

Este método carga pseudo código procedente de un archivo o una cadena de texto. Si el parametro se trata de un fichero, se cargará todo el código que contenga, en caso contrario se procesará el parametro como cadena de texto que incluye el código a procesar.

Debe ser usado en conjunción con el método SavePCode().

Tipo	Estándar
Parámetros	<**cText	cFile**>: Texto o archivo a procesar
Valor de retorno	<**lSuccess**>: Verdadero si éxito

1.7.1.2.7 TScript:MsgError

Muestra un mensaje de error con el último error producido. Igualmente retorna verdadero si existe algún error.

Tipo	Estándar
Parámetros	Ninguno
Valor de retorno	<**lError**>: Verdadero si error

1.7.1.2.8 TScript:New

Constructor de la clase. Admite como único parametro el texto a compilar.

Tipo	Constructor
Parámetros	[<**cText**>]: Texto a compilar
Valor de retorno	<**Self**>: Referencia al objeto

1.7.1.2.9 TScript:Reset

Borra todas la funciones y variables de TScript.

Tipo	Estándar
Parámetros	Ninguno
Valor de retorno	NIL

1.7.1.2.10 TScript:Run

Ejecuta la función 'cFunction' con hasta diez parametros (p1 a p10).

Tipo	Estándar
Parámetros	<**cFunction**>: Función a ejecutar [<**p1**>1,...,<**p10**>]: Parámetros de la función
Valor de retorno	<**xRet**>: Valor retornado por la función

1.7.1.2.11 TScript:SavePCode

Guarda como pseudo código todas las funciones existentes en TScript.

Cuando el parametro **cFile** no es pasado el método retorna como una cadena de caracteres todo el pseudo código. Si **cFile** es pasado como parametro el pseudo código es grabado en el ficheo y el método retorna un valor lógico indicando el éxito de la operación.

Debe ser usado en conjunción con el método LoadPCode().

Tipo	Estándar
Parámetros	[<**cFile**>]: Nombre del fichero.
Valor de retorno	<**xReturn**>: Un valor lógico indicando el éxito de la operación si se paso el parametro cFile. En caso contrario

	retorna el propio pseudo código como una cadena de caracteres.

1.7.2 TScrFun

Esta clase encapsula cada función existente en el script. Usted no requerirá normalmente acceder a esta clase ya que puede acceder a ella directamente desde la clase TScript.

1.7.2.1 TScrFun:Data

■ Sólo lectura ■ Asignable

Ámbito	Nombre	Tipo	Valor inicial
■	aLines	Array	{}
■	cName	Carácter	""
■	oScript	Objeto	.F.
■	xReturn	Any	NIL

1.7.2.1.1 TScrFun:aLines

Matriz con todos los objetosl TScrLin de la función. Esta matriz se rellena en el proceso de compilación.

Ámbito:	Sólo lectura
Tipo:	Matriz
Valor inicial:	{}

1.7.2.1.2 TScrFun:cName

Nombre de la función.

Ámbito:	Sólo lectura
Tipo:	Carácter
Valor inicial:	""

1.7.2.1.3 TScrFun:oScript

Referencia a su contenedor TScript.

Ámbito:	Sólo lectura
Tipo:	Objeto
Valor inicial:	NIL

1.7.2.1.4 TScrFun:xReturn

Valor retornado por la función.

Ámbito:	Sólo lectura
Tipo:	Cualquiera
Valor inicial:	NIL

1.7.2.2 TScrFun:Métodos

Constructor Estándar

Tipo	Nombre
	Call
	Run

1.7.2.2.1 TScrFun:Call

Ejecuta la función sin crear sus variables. Ver explicación de Run().

Tipo	Estándar
Parámetros	Ninguno
Valor de retorno	**<xReturn>**: Valor retornado por la función

1.7.2.2.2 TScrFun:Run

Este método ejecuta la función con lo parametros **aParams**. Si el parametro **bCall** es pasado, entonces sólo las variables son incializadas siendo responsable el code-block **bCall** de ejecutar el método Call() para que empiece la ejecución. El uso de **bCall** se puede utilizar par ganar algo de velocidad debido a la continua creación de variables que realiza el método **Run**().

Tipo	Estándar

Parámetros	[<**aParams**>]: Matriz con los parametros de la función [<**bCall**>]: Code block que llamará al método Call
Valor de retorno	<**xReturn**>: Valor retornado por la función

Por ejemplo, si desea ejecutar una función de un script para todos los registros de una base de datos podría realizar algo así:

```
Do while !Eof()
  oScript:Run( "test", 1, 2, 3 )
  skip
End do
```

Pero este código es mucho más rápido:

```
Local oFunction

  oFunction := oScript:GetFunction( "test" )
  oFunction:Run( { 1, 2, 3}, {|oFun| CallFun( oFun ) } )

Function CallFun( oFun )

  Do while !Eof()
    oFun:Call()
    Skip
  End do

return nil
```

1.7.3 TScrLin

Esta clase encapsula cada línea compilada de un objeto TScrFun. Normalmente no necesitará acceder par nada a esta clase.

1.7.3.1 TScrLin:Data

■ Sólo lectura ■ Asignable

Ámbito	Nombre	Tipo	Valor inicial
■	cLine	Carácter	""
■	nLine	Numérico	0
■	oFunction	Objeto	NIL
■	oScript	Objeto	NIL

1.7.3.1.1 TScrLin:cLine

Texto de la línea.

Ámbito:	Sólo lectura
Tipo:	Carácter
Valor inicial:	""

1.7.3.1.2 TScrLin:nLine

Número de la línea.

Ámbito:	Sólo lectura
Tipo:	Numeric
Valor inicial:	0

1.7.3.1.3 TScrLin:oFunction

Referencia a su contenedor TScrFun.

Ámbito:	Sólo lectura
Tipo:	Objeto
Valor inicial:	NIL

1.7.3.1.4 TScrLin:oScript

Referencia a su contendedor TScript.

Ámbito:	Sólo lectura
Tipo:	Objeto
Valor inicial:	NIL

1.7.3.2 TScrLin:Métodos

Ningún método en esta clase.

Índice

- X -