

Harbour Guide

ARRAY()

Crea un array sin inicializar de la longitud especificada

Syntax

```
ARRAY( <nElementos> [, <nElementos>...] ) --> aArray
```

Arguments

<nElementos> es el número de elementos de la dimensión especificada.

Returns

Description

Esta función retorna un array sin inicializar de longitud <nElementos>. Si parámetros <nElementos> adicionales son especificados se crea un array anidado multidimensional sin inicializar dentro de la misma referencia del array. Crear una variable de memoria con el mismo nombre que el array puede destruir el array original y liberar el contenido entero del array. Esto depende, por supuesto del tipo de almacenamiento de ambos: del array y la variable con el mismo nombre que el array.

Examples

- * El siguiente ejemplo crea un array de diez elementos iniciales, luego en cada elemento de ese array, va creando submatrices lineales con la función ARRAY(). Cada una con la misma cantidad de items que la posición que ocupa en aArray. Finalmente lo muestra.

```
LOCAL aArray := Array(10)
LOCAL i := 1, j

FOR i = 1 to LEN( aArray )
  aArray [i] := Array(i)
NEXT

FOR i = 1 to LEN( aArray )
  ? i
  FOR j = 1 to LEN( aArray [i] )
    ?? " ", aArray [i][j]
  NEXT
NEXT
```

Status

Ready

Compliance

Esta función es CA-CLIPPER Compatible en todos los casos, excepto que los arrays en Harbour pueden tener un número ilimitado de elementos mientras que Clipper tiene un límite de 4096 elementos por dimensión. Los arrays en Harbour pueden tener un número ilimitado de dimensiones.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AADD\(\)](#)

[ADEL\(\)](#)

[AFILL\(\)](#)

[AINS\(\)](#)

AADD()

Agrega dinámicamente un nuevo elemento al final de un array

Syntax

```
AADD(<aDestino>, <xValor>) --> Valor
```

Arguments

<aDestino> es el array al cual se agrega un nuevo elemento.

<xValor> es el valor asignado al nuevo elemento.

Returns

AADD() evalúa <xValor> y retorna su valor. Si <xValor> no esta especificado, AADD() retorna NIL.

Description

AADD() es una función que dinámicamente incrementa la longitud actual del array destino en un elemento y asigna el valor <xValor> al recién creado elemento del array. <xValor> puede ser un puntero de referencia a otro array, el cual puede ser asignado a la posición subíndice.

Es útil para construir listas dinámicas o colas (colas). Cada vez que se ejecuta un comando @...GET, el sistema usa AADD() para agregar un nuevo elemento al final del array GetList, y entonces asignar un nuevo objeto Get al nuevo elemento.

Examples

- * Este ejemplo muestra el efecto de múltiples llamadas de la función AADD() a un array, donde va agregando un nuevo elemento cada vez.

```
LOCAL aArray := {}  
FOR x:= 1 to 10  
    AADD( aArray, x)  
NEXT
```

- * Este ejemplo crea un array multidimensional

```
LOCAL aArray := {} // Resultado: aArray es un array vacío  
AADD( aArray, {10, 10122734 }) // Resultado: aArray es {10, 10122734}  
AADD( aArray, {11, 13173645 }) // Resultado: aArray es  
{ { 10, 10122734 }, { 11, 13173645 } }
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas las plataformas

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AINS\(\)](#)

[ARRAY\(\)](#)

ASIZE()

Ajusta (aumenta ó decrementa) el tamaño de un array

Syntax

```
ASIZE(<aDestino>, <nLongitud>) --> aDestino
```

Arguments

<aDestino> es el nombre del array a ser dinámicamente alterado

<nLongitud> es el valor Numérico del nuevo tamaño de <aDestino>

Returns

ASIZE() retorna una referencia al array <aDestino>.

Description

Esta función dinámicamente incrementa ó decrementa el tamaño del array <aDestino> ajustando la longitud del array a <nLongitud> posiciones.

Si la longitud del array <aDestino> is acortada, aquellos elementos al final se pierden. Si la longitud del array es alargada un valor NIL es asignado a los elementos en las nuevas posiciones.

Examples

- * El siguiente ejemplo crea un array con un sólo elemento, luego lo agranda y luego lo vuelve al tamaño original.

```
aArray := { 1 }           // Resultado: aArray es { 1 }
ASIZE( aArray, 3)         // Resultado: aArray es { 1, NIL, NIL }
ASIZE( aArray, 1)         // Resultado: aArray es { 1 }
```

Status

Ready

Compliance

Si HB_COMPAT_C53 es definido, la función genera un Error, de otro modo retornará el mismo array.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AADD\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

ATAIL()

Retorna el último elemento de un array

Syntax

```
ATAIL( <aArray> ) --> Elemento
```

Arguments

<aArray> es el nombre del array a usar

Returns

ATAIL() retorna <Elemento> que puede ser un valor ó una referencia contenida en el último elemento en el array.

Description

Esta función devuelve el último elemento en el array llamado <aArray>. No modifica el tamaño del array ni el valor de ningún subíndice.

Examples

* El siguiente ejemplo crea un array unidimensional y devuelve el último elemento.

```
aArray := { "Cuál", "es el", "futuro", "de xBase ?", "Harbour!" }  
? ATAIL( aArray )
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[ASIZE\(\)](#)

[AADD\(\)](#)

AINS()

Inserta un elemento NIL en una posición del array

Syntax

```
AINS( <aArray>, <nPos> ) --> aDestino
```

Arguments

<aArray> es el nombre del array al que se va a insertar un item

<nPos> es la posición en el <aArray>

Returns

AINS() retorna una referencia al array destino, <aDestino>

Description

Esta función inserta un valor NIL en el array llamado <aArray> en la posición <nPos>.

Todos los elementos del array comenzando con la <nPos> serán desplazados hacia arriba una posición y el último item en el array será removido completamente. En otras palabras, si se va a insertar un item en la quinta posición de un array de diez elementos, el elemento que previamente estaba en la quinta posición ahora será reubicado a la sexta posición. El elemento recién agregado será de tipo NIL y el último elemento es descartado. La longitud del array <aArray> permanece sin cambios.

Examples

- * El siguiente ejemplo crea un array lineal, al cual se inserta un elemento en la quinta posición, perdiéndose el último.

```
LOCAL aArray:= { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }  
AINS( aArray, 5)
```

```
Resultado: aArray es { 1, 2, 3, 4, NIL, 5, 6, 7, 8, 9 }
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AADD\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)

ADEL()

Borra un elemento del array

Syntax

```
ADEL(<aDestino>, <nPos>) --> aDestino
```

Arguments

<aDestino> es el nombre del array cuyo elemento será removido.

<nPos> es la posición del elemento a borrar

Returns

ADEL() retorna una referencia al array destino, <aDestino>

Description

Esta función borra el elemento que se encuentra en la posición <nPos> en el array <aDestino>. Todos los elementos en el array <aDestino> más allá de la posición dada <nPos> serán movidos hacia abajo una posición en el array. En otras palabras, si se borra un item de la quinta posición de un array de diez elementos, el elemento que estaba en la sexta posición ahora será reubicado a la quinta posición. La longitud del array <aDestino> permanece sin cambios y el último elemento en el array toma el valor NIL.

Examples

- * El siguiente ejemplo crea un array lineal, del cual se borra el elemento en la quinta posición.

```
LOCAL aArray:= { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }  
ADEL( aArray, 5)
```

```
Resultado: aArray es { 1, 2, 3, 4, 6, 7, 8, 9, NIL }
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[ACOPY\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

AFILL()

Rellena un array con un valor especificado

Syntax

```
AFILL( <aDestino>, <xValor>, [<nInicio>], [<nContador>] ) --> aDestino
```

Arguments

<aDestino> es el nombre del array a rellenar

<xValor> es la expresión con la que será rellenado <aDestino>

<nInicio> es la posición de comienzo, subíndice del array

<nContador> es el número de elementos que se van a rellenar

Returns

AFILL() retorna una referencia al array destino, <aDestino>

Description

Esta función rellena cada elemento del array llamado <aDestino> con el valor <xValor>. Si es especificado, <nInicio> marca el elemento de inicio para continuar rellenando por <nContador> posiciones. Si no es especificado, el valor de <nInicio> será 1, y el valor de <nContador> será el valor de LEN(<aDestino>); y todos las posiciones del array <aDestino> serán llenadas con la expresión de <xValor>.

Advertencia !: Esta función sólo trabaja en una sola dimensión de <aDestino>. Si hay punteros de referencia a otros arrays dentro de un subíndice de <aDestino> estos valores se perderán, porque esta función los sobrescribe con los nuevos valores.

Examples

- * El siguiente ejemplo crea un array con valores asignados, luego lo rellena con el valor cinco.

```
LOCAL aTest := { NIL, 0, 1, 2 }  
Afill( aTest, 5)           // Resultado aTest es { 5, 5, 5, 5 }
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AADD\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)

ASCAN()

Busca en un array por un valor o hasta que el block devuelva .T.

Syntax

```
ASCAN( <aDestino>, <xBuscar>,  
[<nInicio>], [<nContador>] ) --> nParadoEn
```

Arguments

<aDestino> es el nombre del array a examinar

<xBuscar> es la expresión a encontrar en <aDestino>

<nInicio> es la posición a la cual comenzar la búsqueda

<nContador> es el número de elementos a examinar

Returns

ASCAN() retorna un valor numérico <nParadoEn>, de la posición donde <xBuscar> fué encontrada.

Description

Esta función examina el contenido de un array llamado <aDestino> en busca del valor de <xBuscar>. El valor devuelto es la posición en el array <aDestino> en el cual <xBuscar> fue encontrada. Si esta expresión no es encontrada el valor retornado es cero.

Si es especificada, la posición de inicio al cual comenzar la búsqueda puede ser establecida con el valor pasado en <nInicio>. Por defecto es uno.

Si es especificado, el número de elementos del array a examinar puede ser establecido con el valor pasado en <nContador>. Por defecto es el número total de elementos en el array <aDestino>.

Si <xBuscar> es un bloque de código, la operación de la función es ligeramente diferente. Cada referencia del subíndice del array es pasada al bloque de código para ser evaluada. La rutina de búsqueda continuará hasta que el valor obtenido del bloque de código sea verdadero (.T.) ó hasta que el final del array haya sido alcanzado.

Examples

- * El siguiente ejemplo utiliza una función de biblioteca para llenar el array aDir con los nombres de archivos en el directorio actual. Posteriormente, busca si entre ellos esta presente el archivo test.prg, devuelve cero si no esta, ó mayor de cero si está.

```
LOCAL aDir := DIRECTORY( "*.*)"
? ASCAN( aDir,,,{|x,y| x[1] == "test.prg" } )
```

Status

Ready

Compliance

Esta función no es compatible con CA-Clipper . La función ASCAN() de Clipper es afectada por la condición SET EXACT ON/OFF

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[AEVAL\(\)](#)

[ARRAY\(\)](#)

AEVAL()

Ejecuta un bloque de código por cada elemento en el array

Syntax

```
AEVAL(<aArray>, <bBlock>, [<nInicio>], [<nContador>]) --> aArray
```

Arguments

<aArray> es el array a ser evaluado.

<bBloque> es el bloque de código a evaluar para cada elemento procesado
<nInicio> es el elemento de inicio del array a evaluar.

<nContador> es el número de elementos a procesar desde <nInicio> hasta el final del array <aArray>

Returns

AEVAL() retorna una referencia a <aArray>

Description

Esta función evalúa y procesa los elementos en <aArray>. Un bloque de código pasado como <bBloque> define la operación a ser ejecutada sobre cada elemento del array. Todos los elementos en <aArray> serán evaluados a menos que sea especificada la posición de comienzo en <nInicio> por <nContador> elementos. Por defecto <nInicio> es uno.

Dos parámetros son pasados al bloque de código <bBloque>. Los elementos individuales en el array son el primer parámetro y su posición en el array es el segundo.

AEVAL() no reemplaza al bucle FOR...NEXT para procesar arrays. Si un array es una unidad autónoma, AEVAL() es apropiado. Si el array va a ser alterado ó si los elementos van a ser reevaluados, un bucle FOR...NEXT es más apropiado.

Examples

Status

Ready

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[EVAL\(\)](#)

[ARRAY\(\)](#)

ACOPY()

Copia elementos de un array a otro

Syntax

```
ACOPY( <aOrigen>, <aDestino>, [[<nPosDestino>] )--> aDestino
```

Arguments

<aOrigen> es el array desde el que se copian los elementos.

<aDestino> es el array al que se copian los elementos.

<nInicio> es la posición desde donde se inicia la copia en <aOrigen>. Por defecto es uno. <nContador> es el número de elementos a copiar comenzando en la posición <nInicio>

<nPosDestino> es la posición de inicio en el array <aDestino> hacia donde se copian los elementos. Por defecto es uno.

Returns

ACOPY() retorna una referencia al array <aDestino>

Description

ACOPY() copia elementos desde el array <aOrigen> hacia el array <aDestino>. Esta función copia todo tipo de datos.

Si un elemento en el array <aOrigen> es un puntero de referencia a otro array (submatriz), esa referencia será copiada al array <aDestino> pero no todas las dimensiones serán copiadas de un array al otro. Esto debe ser realizado via función ACLONE().

Note Si el array <aOrigen> es mayor que <aDestino>, los elementos en el array comienzan a ser copiados en <nPosDestino> y continúan copiándose hasta que el final del array <aDestino> es alcanzado, los elementos que sobran en <aOrigen> se descartan. La función ACOPY() no agrega posiciones al array destino, el tamaño del array <aDestino> permanece constante.

Examples

* El ejemplo siguiente copia un array sobre otro.

```
LOCAL nContador := 2, nInicio := 1, aUltimo, aPrimero  
aUltimo := { "HARBOUR", " es el ", "Heredero" }  
aPrimero := { "CLIPPER", " fue el ", "Pionero" }  
ACOPY( aUltimo, aPrimero, nInicio, nContador)
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[ACLONE\(\)](#)

[ADEL\(\)](#)

[AEVAL\(\)](#)

[AFILL\(\)](#)

[AINS\(\)](#)

[ASORT\(\)](#)

ACLONE()

Duplica un array anidado ó multidimensional

Syntax

```
ACLONE( <aOrigen> ) --> aDuplicado
```

Arguments

<aOrigen> es el nombre del array a ser clonado.

Returns

ACLONE() retorna <aDuplicate> un nueva referencia a otro array exactamente igual al original.

Description

Esta función realiza una copia completa del array llamado <aOrigen>. Crea todas las dimensiones en el array <aDestino> que existen en el array original y luego llena cada dimensión con los mismos valores de los elementos en el original. Ambos arrays coexisten como entidades distintas.

Examples

- * El ejemplo siguiente crea un array bidimensional y lo duplica. Se muestra que son copiadas ambas dimensiones.

```
LOCAL aOrigen, aDestino
aOrigen := { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10} }
aDestino := ACLONE( aOrigen )

* primera dimensión
? "Impares son: " // Resultado: es {1, 3, 5, 7, 9}
FOR n := 1 TO LEN( aDestino)
  ?? aDestino [n][1]
NEXT

* segunda dimensión
? "Pares son: " // Resultado: es {2, 4, 6, 8, 10}
FOR n := 1 TO LEN( aDestino)
  ?? aDestino [n][2]
NEXT
```

Status

Ready

Compliance

Clipper retorna NIL si el parámetro no es un array.

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[ACOPY\(\)](#)

[ADEL\(\)](#)

[AINS\(\)](#)

[ASIZE\(\)](#)

ASORT()

Ordena un array

Syntax

```
ASORT( <aDestino>, [<nInicio>], [<nContador>],  
[<bOrden>] ) --> aDestino
```

Arguments

<aDestino> es el nombre del array a ser ordenado.

<nInicio> es el primer elemento para comenzar el ordenamiento. Por defecto es uno.

<nContador> es el número de elementos a ordenar comenzando en la posición <nInicio>. Por defecto son todos los elementos.

<bOrden> es el bloque de código para el orden de ordenamiento, por defecto es en orden ascendente { | x, y | x < y }. El bloque de código debe recibir dos elementos del array como parametros y debe retornar .T. si el orden es el correcto, .F. en caso contrario.

Returns

ASORT() retorna una referencia al reciente array ordenado <aDestino> ó NIL si el parámetro <aDestino> no es un array.

Description

Esta función ordena todo ó parte de un array dado. Si <bOrden> es omitido, la función espera que <aDestino> sea un array unidimensional conteniendo un solo tipo de datos (uno de: Character, Date, Logical, Numeric) y ordena este array en orden ascendente: los caracteres son ordenados por su valor ASCII, las fechas son ordenadas cronologicamente el valor lógico .F. va antes de .T. y los valores numéricos son ordenados por su valor.

Si <bOrden> es especificado este es usado para manejar la forma de ordenamiento. Cada vez que el bloque es evaluado, dos elementos del array son pasados al bloque de código, y el bloque debe retornar un valor lógico que define si esos elementos estan en orden (.T.) ó no (.F.). Usando este bloque se puede ordenar arrays multidimensionales hacer un ordenamiento descendente ó aún (pero para que querria Ud. hacerlo) ordenar un array que contenga diferentes tipo de datos.

Examples

* El siguiente ejemplo ordena valores numericos en orden ascendente

```
ASORT( { 3, 1, 4, 42, 5, 9 } ) // Resultado: { 1, 3, 4, 5, 9, 42 }
```

* El siguiente ejemplo ordena cadenas en orden descendente

```
LOCAL aKeys := { "Ctrl", "Alt", "Delete" }, n  
LOCAL bOrden := { | x, y | UPPER( x ) > UPPER( y ) }  
ASORT( aKeys,,, bOrden )  
FOR n = 1 TO LEN( aKeys )  
  ? aKeys [n] // Resultado: { "Delete", "Ctrl", "Alt" }  
NEXT
```

* El siguiente ejemplo ordena dos arrays bidimensionales de acuerdo al segundo elemento de cada par.

```
LOCAL aPair := { { "Sun",8}, { "Mon",1}, { "Tue",57}, { "Wed",-6 } }  
ASORT( aPair,,, { | x, y | x[2] < y[2] } )  
  
FOR n = 1 TO LEN( aPair )  
  ? aPair [n][1], aPair [n][2]  
NEXT  
// Resultado: { { "Wed",-6}, { "Mon",1}, { "Sun",8}, { "Tue",57 } }
```

Status

Ready

Compliance

La frecuencia de llamada al bloque de código y el orden difiere de Clipper debido a que Harbour usa un algoritmo distinto (más rápido) de ordenamiento (quicksort).

Files

El código fuente está en arrays.c La librería asociada es vm

See Also:

[ASCAN\(\)](#)

[EVAL\(\)](#)

[ARRAY\(\)](#)

BIN2W()

Convierte 2 bytes de un entero corto sin signo en un número en Harbour

Syntax

```
BIN2W( <cBuffer> ) --> nNumero
```

Arguments

<cBuffer> es una cadena de caracteres que contiene un entero corto sin signo codificado en 16 bits (byte menos significativo primero). Los primeros dos bytes son tenidos en cuenta, si hay más son ignorados.

Returns

BIN2W() retorna un número entero (ó cero si <cBuffer> no es una cadena).

Description

BIN2W() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. BIN2W() toma dos bytes de un entero corto sin signo codificado en 16 bits y los convierte en un valor numérico estandar de Harbour.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es también una forma útil de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). BIN2W() es la inversa de W2BIN()

Examples

```
// Muestra la longitud de la cabecera de un DBF
FUNCTION main()
LOCAL nHandle, cBuffer := SPACE( 2 )
nHandle := FOPEN( "test.dbf" )

IF nHandle > 0
    FSEEK( nHandle, 8 )
    FREAD( nHandle, @cBuffer, 2 )
    ? "Longitud de la cabecera del DBF en bytes:", BIN2W( cBuffer )
    FCLOSE( nHandle )
ELSE
    ? "No puedo abrir el archivo..."
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2W() funciona exactamente como la función de CA-Clipper, BIN2W()

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)
[BIN2L\(\)](#)
[BIN2U\(\)](#)
[I2BIN\(\)](#)
[L2BIN\(\)](#)
[W2BIN\(\)](#)
[WORD\(\)](#)
[U2BIN\(\)](#)
[ARRAY\(\)](#)

BIN2I()

Convierte 2 bytes de un entero corto con signo en un número en Harbour

Syntax

```
BIN2I( <cBuffer> ) --> nNumero
```

Arguments

<cBuffer> es una cadena de caracteres que contiene un entero corto con signo codificado en 16 bits (byte menos significativo primero). Los primeros dos bytes son tenidos en cuenta, si hay más son ignorados.

Returns

BIN2I() retorna un número entero (ó cero si <cBuffer> no es una cadena).

Description

BIN2I() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. BIN2I() toma dos bytes de un entero corto con signo codificado en 16 bits y los convierte en un valor numérico estándar de Harbour.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma útil de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). BIN2I() es la inversa de I2BIN()

Examples

```
// Muestra la fecha de la última actualización del DBF
FUNCTION main()
LOCAL nHandle, cAno, cMes, cDia
nHandle := FOPEN( "test.dbf" )

IF nHandle > 0
    FSEEK( nHandle, 1 )
    cAno := cMes := cDia := " "
    FREAD( nHandle, @cAno, 1 )
    FREAD( nHandle, @cMes, 1 )
    FREAD( nHandle, @cDia, 1 )
    ? "Fecha de actualización:", BIN2I( cAno ), BIN2I( cMes ), ;
    BIN2I( cDia )
    FCLOSE( nHandle )
ELSE
    ? "No puedo abrir el archivo..."
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2I() funciona exactamente como la función de CA-Clipper, BIN2I()

Files

La librería es rtl

See Also:

[BIN2L\(\)](#)

[BIN2U\(\)](#)

[BIN2W\(\)](#)

[I2BIN\(\)](#)

[L2BIN\(\)](#)

[W2BIN\(\)](#)

[WORD\(\)](#)

U2BIN()
ARRAY()

BIN2L()

Convierte 4 bytes de un entero largo con signo en un número en Harbour

Syntax

```
BIN2L( <cBuffer> ) --> nNumero
```

Arguments

<cBuffer> es una cadena de caracteres que contiene un entero largo con signo codificado en 32 bits (byte menos significativo primero). Los primeros cuatro bytes son tenidos en cuenta, si hay más son ignorados.

Returns

BIN2L() retorna un número entero (ó cero si <cBuffer> no es una cadena).

Description

BIN2L() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. BIN2L() toma cuatro bytes de un entero largo con signo codificado en 32 bits y los convierte en un valor numérico estándar de Harbour.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma útil de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). BIN2L() es la inversa de L2BIN()

Examples

```
// Muestra el número de registros en el DBF
FUNCTION main()
LOCAL nHandle, cBuffer := SPACE( 4 )
nHandle := FOPEN( "test.dbf" )

IF nHandle > 0
    FSEEK( nHandle, 4 )
    FREAD( nHandle, @cBuffer, 4 )
    ? "número de registros en el archivo:", BIN2L( cBuffer )
    FCLOSE( nHandle )
ELSE
    ? "No puedo abrir el archivo..."
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2L() funciona exactamente como la función de CA-Clipper, BIN2L()

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)
[BIN2U\(\)](#)
[BIN2W\(\)](#)
[I2BIN\(\)](#)
[L2BIN\(\)](#)
[W2BIN\(\)](#)
[WORD\(\)](#)
[U2BIN\(\)](#)
[ARRAY\(\)](#)

BIN2U()

Convierte 4 bytes de un entero largo sin signo en un número en Harbour

Syntax

```
BIN2U( <cBuffer> ) --> nNumero
```

Arguments

<cBuffer> es una cadena de caracteres que contiene un entero largo sin signo codificado en 32 bits (byte menos significativo primero). Los primeros cuatro bytes son tenidos en cuenta, si hay más son ignorados.

Returns

BIN2U() retorna un número entero (ó cero si <cBuffer> no es una cadena).

Description

BIN2U() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. BIN2U() toma cuatro bytes de un entero largo sin signo codificado en 32 bits y los convierte en un valor numérico estándar de Harbour.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma util de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). BIN2U() es la inversa de U2BIN()

Examples

```
// Muestra el número de registros en el DBF
FUNCTION main()
LOCAL nHandle, cBuffer := SPACE( 4 )
nHandle := FOPEN( "test.dbf" )

IF nHandle > 0
    FSEEK( nHandle, 4 )
    FREAD( nHandle, @cBuffer, 4 )
    ? "Número de registros en el archivo:", BIN2U( cBuffer )
    FCLOSE( nHandle )
ELSE
    ? "No puedo abrir el archivo..."
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2U() es una función de compatibilidad con XBase++ y no existe como una función estándar en CA-Clipper 5.x Esta función es solamente visible si el archivo source/rtl/binnum.c fue compilado con la bandera HB_COMPAT_XPP.

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)
[BIN2L\(\)](#)
[BIN2W\(\)](#)
[I2BIN\(\)](#)
[L2BIN\(\)](#)
[W2BIN\(\)](#)
[WORD\(\)](#)
[U2BIN\(\)](#)

[ARRAY\(\)](#)

I2BIN()

Convierte un número en Harbour en 2 bytes de un entero corto con signo en

Syntax

```
I2BIN( <nNumero> ) --> cBuffer
```

Arguments

<nNumero> es un valor numérico a convertir (los dígitos decimales son ignorados).

Returns

I2BIN() retorna una cadena de caracteres de dos bytes que contienen un entero corto con signo, codificado en 16 bits (byte menos significativo primero).

Description

I2BIN() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. I2BIN() toma un valor numérico y lo convierte en dos bytes de un entero corto con signo, codificado en 16 bits.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma util de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). I2BIN() es la inversa de BIN2I()

Examples

```
// El ejemplo cambia la "fecha de ultima actualización" del DBF
#include "fileio.ch"
FUNCTION main()
LOCAL nHandle, cAno, cMes, cDia

USE test
? "La fecha original de actualización es:", LUPDATE()
CLOSE
nHandle := FOPEN( "test.dbf", FO_READWRITE )

IF nHandle > 0
    FSEEK( nHandle, 1, )
    cAno := I2BIN( 68 )
    cMes := I2BIN( 8 )
    cDia := I2BIN( 1 )
    FWRITE( nHandle, cAno , 1 ) // escribe solo el primer byte
    FWRITE( nHandle, cMes, 1 )
    FWRITE( nHandle, cDia, 1 )
    FCLOSE( nHandle )
    USE test
    ? "La nueva fecha de actualización es:", LUPDATE()
    CLOSE
ELSE
    ? "No puedo abrir el archivo..."
ENDIF
RETURN NIL
```

Status

Ready

Compliance

I2BIN() funciona exactamente como la función de CA-Clipper, I2BIN()

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)

[BIN2L\(\)](#)
[BIN2U\(\)](#)
[BIN2W\(\)](#)
[L2BIN\(\)](#)
[W2BIN\(\)](#)
[WORD\(\)](#)
[U2BIN\(\)](#)
[ARRAY\(\)](#)

W2BIN()

Convierte un número en Harbour en 2 bytes de un entero corto sin signo en

Syntax

```
W2BIN( <nNumero> ) --> cBuffer
```

Arguments

<nNumero> es un valor numérico a convertir (los dígitos decimales son ignorados).

Returns

W2BIN() retorna una cadena de caracteres de dos bytes que contienen un entero corto sin signo, codificado en 16 bits (byte menos significativo primero).

Description

W2BIN() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. W2BIN() toma un valor numérico y lo convierte en dos bytes de un entero corto sin signo, codificado en 16 bits.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma util de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). W2BIN() es la inversa de BIN2W()

Status

Ready

Compliance

W2BIN() es una función de compatibilidad con XBase++ y no existe como una función estandar en CA-Clipper 5.x Esta función es solamente visible si el archivo source/rtl/binnum.c fue compilado con la bandera HB_COMPAT_XPP.

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)

[BIN2L\(\)](#)

[BIN2U\(\)](#)

[BIN2W\(\)](#)

[I2BIN\(\)](#)

[L2BIN\(\)](#)

[WORD\(\)](#)

[U2BIN\(\)](#)

[ARRAY\(\)](#)

L2BIN()

Convierte un número en Harbour en 4 bytes de un entero largo con signo en

Syntax

```
L2BIN( <nNumero> ) --> cBuffer
```

Arguments

<nNumero> es un valor numérico a convertir (los dígitos decimales son ignorados).

Returns

L2BIN() retorna una cadena de caracteres de cuatro bytes que contienen un entero largo con signo, codificado en 32 bits (byte menos significativo primero).

Description

L2BIN() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. L2BIN() toma un valor numérico y lo convierte en cuatro bytes de un entero largo con signo, codificado en 32 bits.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma útil de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). L2BIN() es la inversa de BIN2L()

Status

Ready

Compliance

L2BIN() funciona exactamente como la función de CA-Clipper, L2BIN()

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)

[BIN2L\(\)](#)

[BIN2U\(\)](#)

[BIN2W\(\)](#)

[I2BIN\(\)](#)

[W2BIN\(\)](#)

[WORD\(\)](#)

[U2BIN\(\)](#)

[ARRAY\(\)](#)

U2BIN()

Convierte un número en Harbour en 4 bytes de un entero largo sin signo en

Syntax

```
U2BIN( <nNumero> ) --> cBuffer
```

Arguments

<nNumero> es un valor numérico a convertir (los dígitos decimales son ignorados).

Returns

U2BIN() retorna una cadena de caracteres de cuatro bytes que contienen un entero largo sin signo, codificado en 32 bits (byte menos significativo primero).

Description

U2BIN() es una de las funciones de conversión binaria, de bajo nivel. Estas funciones convierten entre un valor numérico en Harbour y una representación de cadena de un valor numérico dado. U2BIN() toma un valor numérico y lo convierte en cuatro bytes de un entero largo sin signo, codificado en 32 bits.

Ud podría preguntarse cual es la necesidad de este tipo de funciones, bueno, primero de todo esta permite leer/escribir información desde/ hacia un archivo binario (como extraer información de la cabecera de un archivo DBF), es tambien una forma útil de compartir información desde otra fuente distinta a Harbour (lenguaje C por ejemplo). U2BIN() es la inversa de BIN2U()

Status

Ready

Compliance

U2BIN() es una función de compatibilidad con XBase++ y no existe como una función estándar en CA-Clipper 5.x Esta función es solamente visible si el archivo source/rtl/binnum.c fue compilado con la bandera HB_COMPAT_XPP.

Files

La librería es rtl

See Also:

[BIN2I\(\)](#)

[BIN2L\(\)](#)

[BIN2U\(\)](#)

[BIN2W\(\)](#)

[I2BIN\(\)](#)

[L2BIN\(\)](#)

[W2BIN\(\)](#)

[WORD\(\)](#)

[ARRAY\(\)](#)

WORD()

Convierte parámetros de doble precisión del mandato CALL a enteros.

Syntax

```
WORD( <nDoble> ) --> <nEntero>
```

Arguments

<nDoble> es un valor numérico de doble precisión.

Returns

WORD() retorna un entero en el rango: -32767 a +32767

Description

Esta función convierte valores de doble precisión a enteros, para ser usados con el mandato CALL.

Examples

```
// Utiliza WORD() como argumento del mandato CALL  
CALL INVENT WITH WORD(75300)
```

Status

Ready

Compliance

La NG de Clipper NG establece que WORD() funcionará solamente cuando sea usada en la lista de parámetros del comando CALL, de otra manera devolverá NIL, en Harbour esta funcionará donde sea.

Files

La librería es rtl

See Also:

[ARRAY\(\)](#)

DBEDIT()* Despliega registros en una tabla

Syntax

```
DBEDIT( [<nSup>], [<nIzq>], [<nInf>], [<nDer>], [<acColumnas>], ;  
[<xFuncionUsuario>], [<xColumnSayPictures>], [<xCabeceraColumna>], ;  
[<xSeparadorCabecera>], [<xSeparadorColumna>], ;  
[<xSeparadorPie>], [<xPieColumna>] ) --> lExit
```

Arguments

<nSup> coordenada para la fila Superior de visualización. El rango para <nSup> va de cero a MAXROW(), por defecto es cero.

<nIzq> coordenada para la columna izquierda de visualización. El rango para <nIzq> va de cero a MAXCOL(), por defecto es cero.

<nInf> coordenada para la fila inferior de visualización. El rango para <nInf> va de cero a MAXROW(), por defecto es MAXROW().

<nDer> coordenada para la columna derecha de visualización. El rango para <nDer> va de cero a MAXCOL(), por defecto es MAXCOL().

<acColumnas> es un array de expresiones de caracteres que contienen los nombres de los campos de la base de datos, o expresiones para la visualización en cada columna. Si no es especificada, por defecto es la visualización de todos los campos de la base de datos en el area de trabajo actual.

<xFuncionUsuario> es el nombre de una función ó un bloque de código que podría ser llamado cada vez que una tecla no reconocida ha sido presionada ó cuando no hay más teclas para ser procesadas y DBEDIT() va al modo inactivo. Si <xFuncionUsuario> es una cadena de caracteres, esta debe contener el nombre de una función de usuario definida, que sea válida y sin paréntesis. Ambos la función definida por el usuario ó el bloque de código deberán aceptar dos parámetros: nModo y nActualColumna. Ambos deberían devolver un valor numérico que corresponda a uno de los códigos de retorno esperados. (Vea la tabla más abajo con la lista de nModo y los códigos de retorno)

<xColumnSayPictures> es un molde (picture) opcional. Si <xColumnSayPictures> es una cadena de caracteres, todas las columnas pueden usar el mismo valor como cadena de molde (picture). Si <xColumnSayPictures> es un array, cada elemento debe ser una cadena de caracteres que corresponde a la cadena (picture) usada de molde para la columna con el mismo índice. Vea la ayuda para @...SAY para tener más información acerca de los valores de molde (picture).

<xCabeceraColumna> contiene los títulos de cabecera para cada columna si este es una cadena de caracteres, todas las columnas tendrán el mismo encabezado, si este es un array, cada elemento es una cadena de caracteres que contienen el título de cabecera para cada campo. La cabecera puede ser dividido en más de una línea al poner un punto y coma (;) en los lugares donde Ud. desea romper la línea. Si es omitida el valor por defecto para cada cabecera de columna es tomado de <acColumnas> ó el nombre del campo de la base de datos, si <acColumnas> no fue especificado

<xSeparadorCabecera> es un array que contiene caracteres que dibujan las líneas que separan las cabeceras y los datos de los campos. En lugar de un array Ud. puede usar una cadena de caracteres que podría ser usada para la visualización de la misma línea para todos los campos. El valor por defecto es una línea doble.

<xSeparadorColumna> es un array que contiene caracteres que dibujan las líneas que separan las columnas visualizadas. En lugar de un array Ud. puede usar una cadena de caracteres que podría ser usada para la visualización de la misma línea para todos los campos. El valor por defecto es una línea simple.

<xSeparadorPie> es un array que contiene caracteres que dibujan las líneas que separan el area de datos de los campos y el pie. En lugar de un array Ud. puede usar una cadena de caracteres que podría ser usada para la visualización de la misma línea para todos los campos. El valor por defecto es ningún separador de pie.

<xPieColumna> contiene el pie para ser visualizado al final de cada columna, si este es una cadena de caracteres, todas las columnas tendrán el mismo pie, si este es un array, cada elemento es una cadena de caracteres que contienen el pie para cada campo. El pie puede ser dividido en más de una línea al poner un punto y coma (;) en los lugares donde Ud. desea romper la línea. Si es omitido ningún pie

es visualizado.

Returns

DBEDIT() retorna .F. si no hay una base de datos abierta en ese area de trabajo, ó si el número de columnas para la visualización es cero, en caso contrario DBEDIT() devuelve .T.

Description

DBEDIT() visualiza y permite editar registros de una ó más areas de trabajo en una grilla en pantalla. Cada columna es definida por los elementos de <acColumnas> y es el equivalente de un campo. Cada fila es el equivalente de un registro de la base de datos.

A Continuación estan las teclas manejadas por DBEDIT():

----- <table> Tecla
Significado

Izquierda	
Derecha	
Arriba	
Abajo	
Pag-Arriba	
Pag-Abajo	
Ctrl Pag-Abajo	
Inicio	
Fin	
Ctrl Izquierda	
Ctrl Derecha	
Ctrl Inicio	
Ctrl Fin	

Cuando <xFuncionUsuario> es omitida, dos teclas más estan activas: <table>
Tecla significado

Esc	
Enter	

Cuando DBEDIT() ejecuta <xFuncionUsuario> le pasa los siguientes argumentos: nModo y el índice del registro actual en <acColumnas>. Si <acColumnas> es omitido, el número de índice es número de FIELD() de la estructura de la base de datos abierta.

Valores de los Modos en DBEDIT() :

----- <table> Dbedit.ch Valor Significado

DE_IDLE			
DE_HITTOP			
DE_HITBOTTOM			
DE_EMPTY			
DE_EXCEPT			

La función definida por el usuario ó el bloque de código debe retornar un valor que le indique a DBEDIT() que hacer a continuación.

Codigos de retorno de la Función del Usuario:

La función del usuario es llamada una vez en cada uno de los siguientes casos: - La base de datos esta vacía. - El usuario trata de mover más alla de la parte superior ó de la parte inferior del archivo. - Interrupción de teclado, el usuario ha presionado una tecla que no puede ser manejada por DBEDIT(). - El buffer de teclado esta vacío ó un refresco de pantalla acaba de ocurrir.

Nota Importante: ----- DBEDIT() es una función de compatibilidad, esta fué superada por la clase TBrowse y no es recomendada para nuevas aplicaciones.

Examples

```
// Despliega un archivo DBF usando valores por defecto
USE Test
DBEDIT()
```

Status

Started

Compliance

<xFuncionUsuario> puede ser también un bloque de código, esta es una extensión de Harbour.

CA-Clipper lanzará un error si no hay una base de datos abierta Harbour puede retornar .F.

Clipper es inconsistente y lanzará un error si el número de columnas es cero. Harbour puede retornar .F. Las NG de CA-Clipper 5.2 indican que el valor devuelto es NIL, esto es erróneo y debería ser un valor lógico.

Hay un código de retorno (3) indocumentado para las funciones definidas por el usuario en Clipper (ambos 87 and 5.x). este es un Modo de agregado (Append) el cual: "Divide la pantalla para permitir más que datos sean agregados en el area de la ventana". Este modo no es soportado por Harbour.

Files

Los archivos de cabecera son dbedit.ch, inkey.ch La librería es rtl

See Also:

[ARRAY\(\)](#)
[BROWSE\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)

BROWSE()

Despliega un archivo de base de datos

Syntax

```
BROWSE( [<nSup>, <nIzq>, <nInf>, <nDer>] ) --> lExito
```

Arguments

<nSup> coordenada de la fila Superior de visualización.

<nIzq> coordenada de la columna izquierda de visualización.

<nInf> coordenada de la fila inferior de visualización.

<nDer> coordenada de la columna derecha de visualización.

Returns

BROWSE() retorna .F. si no hay una base de datos abierta en ese area de trabajo, en caso contrario devuelve .T.

Description

BROWSE() es un visualizador de bases de datos de propósito general, sin demasiado trabajo ud. puede desplegar un archivo DBF con las siguientes teclas:

Tecla	Significado
Izquierda	Mueve una columna a la izquierda (campo previo)
Derecha	Mueve una columna a la derecha (proximo campo)
Arriba	Mueve arriba una fila (registro previo)
Abajo	Mueve abajo una fila (proximo registro)
Pag-Arriba	Mueve a la pantalla previa
Pag-Abajo	Mueve a la pantalla proxima
	Ctrl Pag-Arriba Mueve al inicio del archivo
Ctrl Pag-Abajo	Mueve al final del archivo
Inicio	Mueve a la columna visible más a la izquierda
Fin	Mueve a la columna visible más a la derecha
Ctrl Izquierda	Desplaza una columna a la izquierda
Ctrl Derecha	Desplaza una columna a la derecha
Ctrl Inicio	Mueve a la columna más a la izquierda
Ctrl Fin	Mueve a la columna más a la derecha
Esc	Termina el BROWSE()

En la parte superior de la pantalla ud. ve una linea de estado con la siguiente indicación:

<none>	
<new>	
<Deleted>	
<bof>	

Ud. debería pasar las cuatro coordenadas validas, si menos de cuatro parámetros son pasados al BROWSE() las coordenadas por defecto son: 1, 0, MAXROW(), MAXCOL().

Examples

```
// muestra como desplegar una base de datos  
USE Test  
BROWSE()
```

Status

Started

Files

La librería es rtl

See Also:

[DBEDIT\(\)*](#)

[ARRAY\(\)](#)

TBrowseDB()

Crea un nuevo objeto TBrowse para ser usado con una base de datos.

Syntax

```
TBrowseDB( [<nSup>], [<nIzq>], [<nInf>], [<nDer>] ) --> oBrowse
```

Arguments

<nSup> coordenada de la fila superior de visualización.

<nIzq> coordenada de la columna izquierda de visualización.

<nInf> coordenada de la fila inferior de visualización.

<nDer> coordenada de la columna derecha de visualización.

Returns

TBrowseDB() retorna un nuevo objeto TBrowse con unas coordenadas específicas y un :SkipBlock, :GoTopBlock y :GoBottomBlock por defecto para desplegar una base de datos.

Description

TBrowseDB() es una forma rápida de crear un objeto TBrowse junto con el soporte mínimo para desplegar una base de datos. Note que el objeto TBrowse devuelto no contiene objetos TBColumn y Ud. necesita agregar una columna para cada campo por Ud. mismo

Examples

Para un buen ejemplo, mire en el código fuente de la función BROWSE() en el subdirectorio ../source/rtl/browse.prg

Status

Started

Compliance

TBrowseDB() funciona exactamente como la función de CA-Clipper TBrowseDB().

Files

La librería es rtl

See Also:

[BROWSE\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

dbSkipper()

Función para ayudar a saltar registros en la base de datos

Syntax

```
dbSkipper( <nRecs> ) --> nSkipped
```

Arguments

<nRecs> es el número de registros a saltar relativos al registro actual. Números positivos tratan de mover el puntero de registro hacia adelante y Números negativos tratan de mover el puntero de registro hacia atrás <nRecs> registros.

Returns

dbSkipper() retorna el número actual de registros saltados.

Description

dbSkipper() es una función de ayuda usada en el mecanismo de despliegue para saltar un número de registros mientras le da al llamador una indicación del número actual de registros saltados.

Examples

```
// Abre un archivo y chequea si tenemos suficientes registros en el
USE ventas
IF dbSkipper( 100 ) == 100
    ? "Buen trabajo!, debes irte de fiesta"
ELSE
    ? "Muy mal, Ud. debería realmente trabajar más duro"
ENDIF
CLOSE
```

Status

Ready

Compliance

dbSkipper() es una función de compatibilidad con XBase++ y no existe como una función estándar en CA-Clipper 5.x Esta función es solamente visible si el archivo: ../source/rtl/browdb.prg fue compilado con la bandera HB_COMPAT_XPP.

Files

La librería es rtl

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

CLASS

Define una Clase para la Programación Orientada a Objetos (OOP).

Syntax

```
CLASS <NombreClase> [ <FROM, INHERIT> <SuperClase> ]
```

Arguments

<NombreClase> Nombre de la clase a definir. Por tradición, las clases en Harbour comienzan con "T" (algo común en el mundo OOP) para evitar colisiones con clases creadas por el usuario. <SuperClase> La clase padre para usar la herencia.

Description

CLASS crea una clase desde la cual se pueden crear objetos. Cada clase es definida en un archivo.PRG separado para este propósito No se puede crear más de una clase en un archivo.PRG Después del mandato CLASS comienza la definición, luego los elementos DATA (también conocidos como variables de instancia) y luego los METHODS de la clase (el equivalente a las funciones en la programación tradicional)

Las Clases pueden heredar desde una clase sola <SuperClass>, pero la cadena de herencia puede extenderse a muchos niveles

Un program usa una clase llamando al Constructor de la clase, el método New() para crear el objeto. Ese objeto es usualmente asignado a una variable, la cual es usada para acceder a los elementos DATA y a los métodos

Examples

```
CLASS TBColumn

    DATA Block          // Codeblock para recuperar datos para la Columna
    DATA Cargo          // Variable definida por el usuario
    DATA ColorBlock    // Codeblock que determina el color de los items
    DATA ColSep         // Caracter separador de la Columna
    DATA DefColor       // Array de índices numéricos a la tabla de color
    DATA Footing        // Pie de Columna
    DATA FootSep        // Caracter separador del Pie
    DATA Heading        // Encabezado de la Columna
    DATA HeadSep        // Caracter separador de la cabecera
    DATA Width          // Ancho de la Columna
    DATA ColPos         // Posición temporaria de la columna en pantalla

    METHOD New()         // Constructor

ENDCLASS
```

Status

Ready

Compliance

CLASS es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[DATA](#)

[METHOD](#)

DATA

Define una variable de instancia DATA, para los objetos de una clase

Syntax

```
DATA <NombreDato1> [, <NombreDatoN>] [ AS <tipo> ] [ INIT <uValor> ]
```

Arguments

<NombreDato1> Nombre de DATA

<tipo> Especificación Opcional del tipo de datos, de uno de los siguientes (especificación en inglés): Character, Numeric, Date, Logical, Codeblock, Nil

<uValor> Valor opcional inicial cuando se crea un nuevo objeto

Description

Los elementos DATA también pueden ser pensados como "propiedades" ó "atributos" de un objeto. ellos pueden ser de cualquier tipo incluyendo bloques de código. Una vez que el objeto ha sido creado, los elementos DATA son referenciados con dos puntos (:) como en MyObject:Heading := "Nombre". Usualmente una clase también define métodos para manipular los DATA.

Se puede usar la cláusula "AS <type>" para reforzar que DATA es perteneciente a un cierto tipo. De otra manera este tomará el tipo de cualquier valor que le sea asignado.

Use la cláusula "INIT <uValue>" para inicializar ese DATA a <uValue> siempre que un nuevo objeto es creado.

Examples

```
CLASS TBColumn

    DATA Block          // Codeblock para recuperar datos para la Columna
    DATA Cargo          // Variable definida por el usuario
    DATA ColorBlock    // Codeblock que determina el color de los items
    DATA ColSep         // Caracter separador de la Columna
    DATA DefColor       // Array de índices numéricos a la tabla de color
    DATA Footing        // Pie de Columna
    DATA FootSep        // Caracter separador del Pie
    DATA Heading        // Encabezado de la Columna
    DATA HeadSep        // Caracter separador de la cabecera
    DATA Width          // Ancho de la Columna
    DATA ColPos         // Posición temporaria de la columna en pantalla

    METHOD New()         // Constructor

ENDCLASS
```

Status

Ready

Compliance

DATA es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[CLASS](#)

[METHOD](#)

[CLASSDATA](#)

CLASSDATA

Define una variable CLASSDATA para una clase (NO para un Objeto !)

Syntax

```
CLASSDATA <DataName1> [,<DataNameN>] [ AS <type> ] [ INIT <uValue> ]
```

Arguments

<NombreDatol> Nombre de DATA

<tipo> Especificación opcional del tipo de datos de uno de los siguientes (Original en inglés): Character, Numeric, Date, Logical, Codeblock, Nil

<uValor> Valor opcional inicial cuando se inicia el programa

Description

Las variables CLASSDATA pueden ser pensadas como "propiedades" de una clase entera. Cada CLASSDATA existe sólo una vez, no importa cuántos objetos sean creados. Un uso común es para un contador que es incrementado siempre que un objeto es creado y decrementado cuando alguno es destruido, así se puede monitorear el número de objetos en existencia para esta clase.

Se puede usar la cláusula "AS <type>" para reforzar que CLASSDATA es perteneciente a un cierto tipo. De otra manera este tomará el tipo de cualquier valor que le sea asignado. Use la cláusula "INIT <uValue>" para inicializar ese DATA a <uValue> siempre que un nuevo objeto es creado.

Examples

```
CLASS TWindow
  DATA hWnd, nOldProc
  CLASSDATA lRegistered AS LOGICAL
ENDCLASS
```

Status

Ready

Compliance

CLASSDATA es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[CLASS](#)

[METHOD](#)

[DATA](#)

METHOD

Declara un METHOD (método) para una clase en la cabecera de la clase

Syntax

```
METHOD <NombreMétodo>( [<params,...>] ) [ CONSTRUCTOR ]
METHOD <NombreMétodo>( [<params,...>] ) INLINE <Code,...>
METHOD <NombreMétodo>( [<params,...>] ) BLOCK <CodeBlock>
METHOD <NombreMétodo>( [<params,...>] ) EXTERN <FuncName>([<args,...>])
METHOD <NombreMétodo>( [<params,...>] ) SETGET
METHOD <NombreMétodo>( [<params,...>] ) VIRTUAL
METHOD <NombreMétodo>( [<param>] ) OPERATOR <op>
METHOD <NombreMétodo>( [<params,...>] ) CLASS <ClassName>
```

Arguments

<NombreMétodo> Nombre del método a definir
<params,...> Lista opcional de parametros

Description

Los Métodos son "funciones de clase" y hacen el trabajo de la clase. Todos los métodos pueden ser definidos en la cabecera entre los comandos CLASS y ENDCLASS. Si el cuerpo de un método no esta totalmente definido aqui, el cuerpo completo es escrito debajo del comando ENDCLASS usando esta sintaxis:

```
METHOD <NombreMétodo>( [<params,...>] ) CLASS <NombreClase>
```

Los Métodos pueden referenciar al objeto actual usando la palabra clave "Self:" ó su versión más corta "::"

CLAUSULAS:

CONSTRUCTOR Define un método especial de la clase: el método Constructor, usado para crear objetos. Este es usualmente el método New(). Los constructores siempre retornan el objeto New().

INLINE Rápido y fácil de codificar, INLINE le permite definir el código para el método inmediatamente después de la definición de la clase. Cualquier método no declarado INLINE ó BLOCK debe ser completamente definido después del comando ENDCLASS. El *<Codigo,...>* siguiente a INLINE recibe un parametro de Self. Si se necesita recibir más parámetros, use la cláusula BLOCK en su lugar.

BLOCK Use esta cláusula cuando desee declarar rápidos métodos 'inline' que necesiten parámetros. El primer parámetro a *<CodeBlock>* debe ser Self, como en:

```
METHOD <NombreMétodo> BLOCK { |Self,<arg1>,<arg2>,...,<argN>|... }
```

EXTERN Si una función externa hace lo que el método necesita, use esta cláusula para hacer una llamada optimizada a esa función directamente.

SETGET Para datos calculados. El nombre del método puede ser manipulado como un elemento de DATA para establecer (Set) u obtener (Get) un valor.

VIRTUAL Métodos que no hacen nada. Utiles para Clases de Base donde la Clase hija definirá el comportamiento del método, ó cuando Ud. esta creando y probando una Clase.

OPERATOR Operador de Sobrecarga para las Clases. Vea el ejemplo ../Tests/TestOp.prg para detalles

CLASS <ClassName> Use esta sintaxis solamente para definir un método completo después del comando ENDCLASS.

Examples

```
CLASS TWindow
  DATA hWnd, nOldProc
  METHOD New( ) CONSTRUCTOR
  METHOD Capture() INLINE SetCapture( ::hWnd )
  METHOD End() BLOCK { | Self, lEnd | If( lEnd := ::lValid(),;
    ::PostMsg( WM_CLOSE ),), lEnd }
  METHOD EraseBkgnd( hDC )
```

```
METHOD cTitle( cNewTitle ) SETGET
METHOD Close() VIRTUAL
ENDCLASS

METHOD New( ) CLASS TWindow
  local nVar, cStr
  ... <codigo> ...
  ... <codigo> ...
RETURN Self
```

Tests

TestOp.prg

Status

Ready

Compliance

METHOD es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[DATA](#)

[CLASS](#)

MESSAGE

Reenvía la llamada a un método a otro método.

Syntax

```
MESSAGE <NombreMensaje> METHOD <NombreMétodo>( [<params,...>] )  
MESSAGE <NombreMensaje>() METHOD <NombreMétodo>( [<params,...>] )
```

Arguments

<NombreMensaje> El nombre del pseudo-método a definir.

<NombreMétodo> El método a crear y llamar cuando <NombreMensaje> es
invocado. <params,...> Lista de parámetros opcionales para el método

Description

El comando MESSAGE es una característica rara vez usada, que permite re-enviar una llamada a un método con un nombre diferente. Esto puede ser necesario si el nombre de un método entra en conflicto con una función pública que necesita ser llamada desde adentro de un método de la Clase.

Por ejemplo, se puede tener una función pública llamada BeginPaint() que es usada para pintar las ventanas. Podría ser natural también tener un método de la Clase Ventana, llamado :BeginPaint() que la aplicación pueda llamar. Pero dentro del método de la Clase uno podría no estar habilitado para llamar a la función pública porque, los métodos internos están basados en funciones estáticas (las que ocultan funciones públicas con el mismo nombre)

El comando MESSAGE permite crear el verdadero método con un nombre diferente (:xBeginPaint()), y aún permitir la sintaxis :BeginPaint() para llamar a :xBeginPaint(). Este entonces es libre de llamar a la función pública BeginPaint().

Examples

```
CLASS TWindow  
  DATA hWnd, nOldProc  
  METHOD New( ) CONSTRUCTOR  
  MESSAGE BeginPaint METHOD xBeginPaint()  
ENDCLASS
```

Status

Ready

Compliance

MESSAGE es una extensión de Harbour.

Platforms

Todas

See Also:

[METHOD](#)

[DATA](#)

[CLASS](#)

[ARRAY\(\)](#)

ERROR HANDLER

Designa un método como manejador de error para la Clase.

Syntax

```
ERROR HANDLER <NombreMétodo>( [<params,...>] )
```

Arguments

<NombreMétodo> Nombre del método a definir

<params,...> Lista de parametros opcionales

Description

ERROR HANDLER nombra al método que debería manejar los errores para la Clase que esta siendo definida.

Examples

```
CLASS TWindow
  ERROR HANDLER MyErrorHandler()
ENDCLASS
```

Status

Ready

Compliance

ERROR HANDLER es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)
[ON_ERROR](#)
[CLASS](#)
[METHOD](#)
[DATA](#)

ON ERROR

Designa un método como manejador de error para la Clase.

Syntax

```
ON ERROR <NombreMétodo>( [<params,...>] )
```

Arguments

<NombreMétodo> Nombre del método a definir

<params,...> Lista de parametros opcionales

Description

ON ERROR es un sinónimo para ERROR HANDLER. Este nombra al método que debería manejar los errores para la Clase que esta siendo definida.

Examples

```
CLASS TWindow
  ON ERROR MyErrorHandler()
ENDCLASS
```

Status

Ready

Compliance

ON ERROR es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[ERROR HANDLER](#)

[CLASS](#)

[METHOD](#)

[DATA](#)

ENDCLASS

Termina la declaración de una Clase.

Syntax

```
ENDCLASS
```

Description

ENDCLASS marca el fin de la declaración de una Clase. Este es usualmente seguido por el método de la Clase que no es INLINE.

Examples

```
CLASS TWindow
  DATA  hWnd, nOldProc
ENDCLASS
```

Status

Ready

Compliance

ON ERROR es una extensión de Harbour.

Platforms

Todas

See Also:

[ARRAY\(\)](#)

[CLASS](#)

[METHOD](#)

[DATA](#)

Opciones del Compilador

Opciones del Compilador

Description

Invocando al compilador Harbour:
=====

```
harbour <archivo[.prg]> [opciones]
or
harbour [opciones] <archivo[.prg]>
or
harbour [opciones] <archivo[.prg]> [opciones]
```

Las opciones de la línea de comandos debe ser separada, al menos por un espacio en blanco.

Las opciones pueden comenzar con el carácter '/' ó '-',

Opciones de la línea de comandos de:
=====

/a Declaración Automática de memvar

Esto causa que todas las variables declaradas por las sentencias PARAMETER, PRIVATE ó PUBLIC sean automáticamente declaradas como variables MEMVAR.

/b Información de depuración (Bug)

El compilador genera toda la información requerida para depuración

/d<id>[=<val>] #define <id>

/es<nivel> establece la Severidad de la salida (Exit Severity)

/es or /es0 - Todas las advertencias son ignoradas y el código de salida retornado por el compilador (accedido por el comando de DOS ERRORLEVEL) es igual a cero si no hay errores en el archivo fuente compilado.

/es1 - Cualquier advertencia genera un código de salida distinto de cero, pero la salida es aún creada.

/es2 - Cualquier advertencia es tratada como error y ningún archivo de salida es creado. El código de salida es establecido a un valor distinto de cero.

/g<tipo> Generación del tipo de archivo de salida <type>

/gc tipo de salida: fuente de lenguaje C (.c) (defecto)

/gf tipo de salida: Windows/DOS OBJ32 (.obj)

/gh tipo de salida: Objeto Portable de Harbour (.hrb)

/gj tipo de salida: fuente de Java (.java)

/gp tipo de salida: fuente de Pascal (.pas)

/gr tipo de salida: recursos de Windows (.rc)

/i<ruta> agrega la ruta de búsqueda de archivos #Include

/l suprime la información del número de Línea

El compilador no genera el número de línea del código fuente en el archivo de salida. La función PROCLINE() retornará cero para los módulos compilados usando esta opción.

/m compilar el Módulo actual solamente

/n sin procedimiento de inicio implícito

El compilador no crea un procedimiento con el mismo nombre que el del archivo compilado. Esto significa que algunas declaraciones puestas antes de la primera sentencia de PROCEDURE ó FUNCTION tienen el alcance del archivo y pueden ser accedidas/usadas en todas las funciones/procedimientos definidos en el archivo

fuente compilado. Todas las sentencias ejecutables puestas al principio del archivo y antes de la primera sentencia de PROCEDURE ó FUNCTION son ignoradas.

`/o<ruta>` unidad de disco y/o ruta para el archivo de salida

`/p` genera un archivo de salida Pre-procesada (.ppo)

El compilador solamente crea el archivo que contiene el resultado del archivo fuente pre-procesado.

`/q` Quieto

El compilador no imprime ningún mensaje durante la compilación (excepto la información del copyright).

`/q0` que permanezca realmente Quieto y no muestre ni siquiera la información del copyright.

`/r[<libreria>]` solicita al linker Revisar por <libreria> (ó ninguna)

Actualmente no soportado en Harbour.

`/s` solo chequeo de Sintaxis.

El compilador chequea la sintaxis solamente. Ningún archivo de salida es generado.

`/t<ruta>` ruta para la creación de un archivo Temporario

Actualmente no usado en harbour (El compilador de harbour no crea ningún archivo temporal)

`/u[<archivo>]` Usar la definición de comando establecido en el <archivo> (ó ninguno)

Aún no soportado.

`/v` las Variables son asumidas como M->

Todas las variables sin declarar ó unaliased son asumidas como variables MEMVAR (variables privadas ó públicas). si este switch no es usado entonces el alcance de estas variables es chequeado en tiempo de ejecución.

`/w[<nivel>]` Establece el número de nivel de las advertencias (Warnings) (0..4, por defecto es 1)

`/w0` - sin advertencias

`/w or /w1` - advertencias compatibles con Clipper

`/w2` - algunas advertencias útiles ausentes en Clipper

`/w3` - advertencias generadas para extensiones al lenguaje hechas

en Harbour. También habilita el chequeo de sintaxis fuertemente tipeada pero sólo advierte contra los tipos declarados, o los tipos que pueden ser calculados en tiempo de compilación.

`/w4` - habilita advertencias acerca de operaciones que son sospechosas, lo cual significa que si se mezclan tipos sin declarar ó tipos que no pueden ser calculados en tiempo de compilación, junto con los tipos ya declarados, una advertencia será generada.

`/x[<prefijo>]` establece el símbolo del prefijo agregado al nombre de función (para archivo.c solamente)

Establece el símbolo del prefijo agregado al nombre de función de inicio (en la salida de lenguaje C, actualmente). Esta función es generada automáticamente para cada módulo de PRG compilado. Este prefijo adicional puede ser usado para suprimir problemas con símbolos duplicados durante el enlazado de una aplicación (linking) con alguna librería de terceros.

`/y` seguimiento de la actividad de Lex & Yacc

El compilador Harbour usa las utilidades FLEX y YACC para analizar el código fuente y generar el archivo de salida requerido. Esta opción sigue la actividad de esas utilidades.

`/z` suprime el cortocircuito lógico (.and. y .or.)

/10 restringe la longitud de símbolos a 10 caracteres.

Todos los nombres de variables y de funciones son cortados a un máximo de 10 caracteres.

Compilación en modo lote (batch).

=====

@<archivo> compila la lista de módulos en el <archivo>

No soportado aún.

Conocidas incompatibilidades entre compiladores harbour y clipper

=====

NOTE:

Si desea librerías de compilación y ejecución 100 % compatibles, entonces Ud. debe definir: HARBOUR_STRICT_CLIPPER_COMPATIBILITY. Esta opción debe ser definida en el archivo ../include/hbsetup.h (en efecto esta opción es puesta en un comentario por defecto - Ud. necesita remover los caracteres /* */ solamente. Este cambio debe ser realizado antes de invocar la utilidad make.

Manejo de variables sin declarar

Cuando un valor es asignado a una variable no declarada y la opción -v de la línea de comandos no es usada, entonces el compilador Clipper asume que la variable es una variable PRIVATE ó PUBLIC y genera un opcode POPM (pop memvar).

Cuando el valor de una variable no declarada es accedido y la opción -v de la línea de comandos no es usada, el compilador Harbour genera un opcode PUSHV (push variable) para determinar el tipo de variable en tiempo de ejecución

Si un campo con el nombre requerido existe en el area de trabajo actual, entonces este valor es usado. Si no existe el campo, entonces una variable PRIVATE ó PUBLIC es usada (si existe).

El compilador Harbour genera un opcode para determinar el tipo de variable en tiempo de ejecución (POPVARIABLE or PUSHVARIABLE) en ambos casos (asignación y acceso).

La diferencia puede ser chequeada por el siguiente código:

```
PROCEDURE MAIN()
```

```
PRIVATE myname
```

```
    DECREATE( "TEST", { { "MYNAME", "C", 10, 0 } } )
```

```
    USE test NEW
```

```
    SELECT test
```

```
    APPEND BLANK
```

```
    FIELD->myname := "FIELD"
```

```
    MEMVAR->myname := "MEMVAR"
```

```
    myname := myname + " assigned"
```

```
    // In Clipper: "FIELD", In Harbour: "FIELD assigned"
```

```
    ? FIELD->myname
```

```
    // In Clipper: "MEMVAR assigned", In Harbour: "MEMVAR"
```

```
    ? MEMVAR->myname
```

```
    USE
```

```
RETURN
```

Pasando por refererencia una variable no declarada

El compilador Harbour usa un opcode especial PUSHV para pasar una

referencia a una variable no declarada (el operador '@'). El tipo de la variable pasada es chequeada en tiempo de ejecución (field or memvar). Sin embargo las variables de campo no pueden ser pasadas por referencia. Esto significa que Clipper chequea sólo la variable memvar y no mira por una de campo. Esta es la razón por la cual el compilador Harbour usa el opcode habitual PUSHMEMVARREF en estos casos. Nótese que el comportamiento en tiempo de ejecución es el mismo en Clipper y en Harbour - sólo los opcodes generados son diferentes.

Manejo de mensajes a objetos

El seteo de HARBOUR_STRICT_CLIPPER_COMPATIBILITY determina la forma en que el envío encadenado de mensajes es manejado

Por ejemplo, el siguiente código:

```
a:b( COUNT() ):c += 1
```

será manejado como:

```
a:b( COUNT() ):c := a:b( COUNT() ):c + 1
en modo de compatibilidad estricta y
```

```
temp := a:b( COUNT() ), temp:c += 1
en modo no-estricto.
```

En la práctica, Clipper llamará a la función COUNT() dos veces: La primera vez antes de la adición y la segunda después de la adición. En Harbour, COUNT() será llamada sólo una vez, antes de la adición.

El método Harbour (no-estricto) es:

- 1) Más rápido
- 2) Garantiza que la misma variable de instancia del mismo objeto será cambiada.

(Ver también: ../source/compiler/expropt.c)

Inicialización variables estáticas

Hay una diferencia en la inicialización de las variables estáticas que son inicializadas con un bloque de código que refiere a una variable local. Por ejemplo:

```
PROCEDURE TEST()
LOCAL MyLocalVar
STATIC MyStaticVar := {|| MyLocalVar }

  MyLocalVar :=0
  ? EVAL( MyStaticVar )

RETURN
```

El código de arriba compila bien en Clipper, pero éste genera un error de ejecución: Error/BASE 1132 Bound error: array access
Called from (b)STATICS\$(0)

En Harbour este código genera un error en tiempo de compilación: Error E0009
Illegal variable (b) initializer: 'MyLocalVar'

Ambos Clipper y Harbour estan manejando todas las variables locales usadas en una forma especial: ellas son separadas de la pila (stack) local de la función / procedimiento donde ellas son declaradas. Esto permite acceder a estas variables despues de la salida de una función / procedimiento. Sin embargo todas las variables estáticas son inicializadas en un procedimiento separado ('STATICS\$' en Clipper y '(_INITSTATICS)' en Harbour) antes del procedimiento principal y antes de todos los procedimientos INIT. Las variables locales no existen en la pila de evaluación (eval stack) donde las variables estáticas son inicializadas, así ellas no pueden ser separadas.

CDOW()

Convierte una fecha al Nombre del día de la semana

Syntax

```
CDOW(<dFecha>) --> cDia
```

Arguments

<dFecha> Cualquier expresión de fecha.

Returns

<cDia> El día actual de la semana.

Description

Esta función devuelve una cadena de caracteres con el día de la semana de una expresión de fecha <dFecha> pasada (DOW=day of week en inglés). Si una fecha NULL es pasada a la función, el valor de la función será un byte NULL.

Examples

```
? CDOW( DATE() )  
  
IF CDOW( DATE() + 10 ) == "SUNDAY"  
? "Este es un día para el descanso."  
ENDIF
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[DAY\(\)](#)

[DOW\(\)](#)

[DATE\(\)](#)

[CMONTH\(\)](#)

CMONTH()

Retorna una cadena de caracteres con el nombre del mes

Syntax

```
CMONTH(<dFecha>) --> cMes
```

Arguments

<dFecha> Cualquier expresión de fecha.

Returns

<cMes> el nombre actual del mes

Description

Esta función retorna el nombre del mes (Enero, Febrero, etc.) de una expresión de fecha <dFecha> pasada a ella. Si una fecha NULL es pasada a la función, el valor de la función será un byte NULL. Nota: El valor devuelto depende del módulo de lenguaje en uso.

Examples

```
? CMONTH( DATE() )           // resultado: Noviembre

IF CMONTH( DATE() + 35) == "Diciembre"
? "Ha hecho compras para las fiestas ?"
ENDIF
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[CDOW\(\)](#)

[DATE\(\)](#)

[MONTH\(\)](#)

[YEAR\(\)](#)

[DOW\(\)](#)

[DTC\(\)](#)

DATE()

Retorna la fecha actual del sistema operativo

Syntax

```
DATE() --> dFechaActual
```

Arguments

Returns

<dFechaActual>, la fecha actual del sistema.

Description

Esta función devuelve la fecha actual del sistema.

Examples

```
? DATE()
```

Tests

```
? "Hoy es ",DAY( DATE())," de ",CMONTH( DATE())," de ",YEAR( DATE())
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[CTOD\(\)](#)

[DTOS\(\)](#)

[DTCO\(\)](#)

[DAY\(\)](#)

[MONTH\(\)](#)

[CMONTH\(\)](#)

CTOD()

Convierte una cadena de caracteres a una expresión de fecha

Syntax

```
CTOD(<cCadenaFecha>) --> dFecha
```

Arguments

<cCadenaFecha> Un fecha en el formato 'mm/dd/yy'

Returns

<dFecha> Una expresión de fecha.

Description

Esta función convierte una fecha que ha sido aportada como una expresión de caracteres a una expresión de fecha. La expresión de caracteres sera en la forma MM/DD/YY" (basada en el valor por defecto en SET DATE) o en el formato apropiado especificado por el comando SET DATE TO. Si una impropia cadena de caracteres es pasada a la función, un valor de fecha vacía será retornado.

Examples

```
? CTOD('12/21/00')
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[ARRAY\(\)](#)

[DATE\(\)](#)

[DTOS\(\)](#)

DAY()

Retorna el número de día del mes en el rango de 0 a 31

Syntax

```
DAY(<cFecha>) --> nMes
```

Arguments

<cFecha> Cualquier expresión válida de fecha.

Returns

Description

Esta función retorna el valor numérico del día del mes de una fecha.

Examples

```
? DAY( CTOD("06/06/1944") )      // Resultado: 6  
? DAY( DATE() + 6325)
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[CTOD\(\)](#)

[DTOS\(\)](#)

[DTCO\(\)](#)

[DATE\(\)](#)

[MONTH\(\)](#)

[CMONTH\(\)](#)

DAYS()

Convierte los segundos transcurridos a días.

Syntax

```
DAYS(<nSegundos> ) --> nDia
```

Arguments

<nSegundos> el número de segundos.

Returns

Description

Esta función convierte <nSegundos> al número equivalente de días; 86399 segundos representan un día, cero segundos es medianoche.

Examples

```
? DAYS( 2434234)
? "Han transcurrido ", DAYS(63251)
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper. Nota: No esta documentada en las NG, ni en el manual en español de CA-Clipper

Platforms

Todas

Files

La librería es rtl

See Also:

[SECONDS\(\)](#)

[SECS\(\)](#)

[ELAPTIME\(\)](#)

DOW()

Convierte una fecha al número del día de la semana (1-7)

Syntax

```
DOW(<dFecha>) --> nDia
```

Arguments

<dFecha> Cualquier expresión válida de fecha.

Returns

DOW() retorna el día de la semana en la forma numérica <nDay>

Description

Esta función convierte un valor de fecha en un número representando el día de la semana. En el rango de 1 al 7, siendo 1 el día Domingo y 7 el día Sábado.

Examples

```
? DOW( DATE() )  
? DOW( DATE() - 6584 )
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[DTCO\(\)](#)

[CDOW\(\)](#)

[DATE\(\)](#)

[DTOS\(\)](#)

[DAY\(\)](#)

DTOC()

Convierte un valor de fecha a una cadena de caracteres

Syntax

```
DTOC(<dFecha>) --> cFecha
```

Arguments

<dFechaString> Cualquier expresión válida de fecha.

Returns

Description

Esta función convierte una expresión de fecha (sea un campo ó una variable) expresada como <dFecha> a una expresión de caracteres en el formato por defecto "MM/DD/YY". El formato de fecha devuelto por esta función es controlado por el formato especificado por el comando SET DATE y es variable a diferencia del devuelto por la función DTOS().

Examples

```
? DTOC( DATE())
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[ARRAY\(\)](#)

[DATE\(\)](#)

[DTOS\(\)](#)

DTOS()

Convierte un valor de fecha a una cadena en el formato AAAAMMDD

Syntax

```
DTOS(<dFecha>) --> cFecha
```

Arguments

<dFecha> Cualquier expresión válida de fecha.

Returns

DTOS() retorna <cFecha>

Description

Esta función convierte un valor de fecha a una cadena de 8 caracteres en el formato AAAAMMDD (no contiene separadores). Si el valor de <dFecha> es una fecha vacía, esta función retorna una cadena de caracteres de 8 espacios en blanco.

Examples

```
? DTOS( DATE())
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[DTCO\(\)](#)

[DATE\(\)](#)

[DTOS\(\)](#)

ELAPTIME()

Calcula el tiempo transcurrido.

Syntax

```
ELAPTIME(<cHoraInicial>,<cHoraFinal>) --> cTranscurrido
```

Arguments

<cHoraInicial> Hora de inicio en el formato de cadena HH:MM:SS <cHoraFinal>
Hora de finalización en el formato de cadena HH:MM:SS

Returns

<cTranscurrido> Diferencia entre tiempos

Description

Esta función devuelve una cadena que muestra la diferencia entre la hora inicial representada por <cHoraInicial> y la hora final representada por <cHoraFinal>. Si la hora de inicio es mayor que la de finalización, la función asume que la fecha a cambiado una vez.

Examples

```
// Muestra el tiempo que se uso el programa parecido a como lo
// hacía el viejo Norton Utilities
STATIC cHoraIni // al inicio del programa
cHoraIni := TIME()

// A la salida del programa
? "Ud. ha usado el programa durante: ", ELAPTIME( cHoraIni, TIME())
QUIT
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[SECS\(\)](#)

[SECONDS\(\)](#)

[TIME\(\)](#)

[DAY\(\)](#)

MONTH()

Extrae el mes como valor numérico de un valor de fecha dado

Syntax

`MONTH(<dFecha>) --> nMes`

Arguments

`<dFecha>` Cualquier expresión válida de fecha.

Returns

`MONTH()` retorna `<nMes>`, número correspondiente al mes del año, en el rango de 0 a 12.

Description

Esta función devuelve el valor numérico del mes contenido en `<dFecha>`. Este valor está en el formato de cuatro dígitos y no es afectado por el seteo de los comandos `SET CENTURY` y `SET DATE`. Si se pasa una fecha vacía `CTOD("")` a esta función devuelve el valor cero.

Examples

? Month(DATE())

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es `rtl`

See Also:

[CDOW\(\)](#)

[DOW\(\)](#)

[YEAR\(\)](#)

[CMONTH\(\)](#)

SECONDS ()

Retorna el número de segundos transcurridos desde medianoche.

Syntax

```
SECONDS() --> nSegundos
```

Arguments

Returns

SECONDS() retorna <nSegundos> desde la medianoche.

Description

Esta función devuelve un valor numérico representando el número de segundos transcurridos desde la medianoche del día actual, basado en la hora actual del sistema operativo. SECONDS() se inicializa en cero (medianoche) y continua hasta 86399 segundos (23:59:59). El valor que devuelve es expresado como segundos y centésimas de segundo.

Examples

```
// Muestra el tiempo que estuvo en ejecución una rutina
nArranque = SECONDS()

// Aca va la llamada a la rutina
// : :
? "Tardó: ", SECONDS()- nArranque , " segundos"
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[TIME\(\)](#)

SECS()

Retorna el número de segundos de la hora del sistema.

Syntax

```
SECS( <cHora> ) --> nSegundos
```

Arguments

<cHora> Expresión de caracteres en el formato de tiempo HH:MM:SS

Returns

SECS() retorna <nsegundos> Numero de segundos

Description

Esta función devuelve un valor numérico que es el número de segundos transcurridos desde medianoche, basados en una cadena de hora dado como <cHora>.

Examples

```
? SECS( TIME() )
? SECS( TIME() - 10)
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper. Nota: No esta documentada en las NG, ni en el manual en español de CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[SECONDS\(\)](#)

[ELAPTIME\(\)](#)

[TIME\(\)](#)

TIME()

Retorna la hora del sistema como una cadena de caracteres

Syntax

```
TIME() --> cHora
```

Arguments

Returns

TIME() retorna la cadena de caracteres <cHora> representando la hora

Description

Esta función devuelve la hora del sistema representada como una expresión de caracteres en el formato HH:MM:SS, para calculos de tiempo conviene usar SECONDS().

Examples

```
? TIME() // Resultado: 21:34:12
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[DATE\(\)](#)

[SECONDS\(\)](#)

YEAR()

Extrae el año como valor numérico de un valor de fecha dado

Syntax

```
YEAR(<dFecha>) --> nAño
```

Arguments

<dFecha> Cualquier expresión válida de fecha.

Returns

YEAR() retorna <nAño> la porción de año de una fecha.

Description

Esta función devuelve el valor numérico del año contenido en <dFecha> Este valor esta en el formato de cuatro dígitos y no es afectado por el seteo de los comandos SET CENTURY y SET DATE. Si se pasa una fecha vacía CTOD("") a esta función devuelve el valor cero.

Examples

```
? YEAR( DATE()) // Resultado: 2000
? YEAR( CTOD("01/31/1982")) // Resultado: 1982
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Platforms

Todas

Files

La librería es rtl

See Also:

[DAY\(\)](#)

[MONTH\(\)](#)

__Dir()*

Muestra por pantalla el listado de archivos.

Syntax

```
__Dir( [<cFileMask>] ) --> NIL
```

Arguments

<cFileMask> Máscara de archivos para incluir en el retorno de la función. Esta podría contener subdirectorios (path) y caracteres estándar usados como comodines, según sean soportados por el sistema operativo (como * y ?). Si <cFileMask> no contiene la ruta al archivo entonces SET DEFAULT es usado para mostrar archivos en la máscara.

Returns

__Dir() siempre retorna NIL.

Description

Si ninguna <cFileMask> es dada, __Dir() muestra información acerca de todos los *.DBF en la ruta SET DEFAULT. Esta información contiene: - Nombre del archivo - Numero de registros - Fecha de la última actualización - Tamaño de cada archivo.

Si <cFileMask> es dada, __Dir() lista todos los archivos que coinciden con la máscara en los siguientes detalles: Nombre, Extensión, Tamaño, Fecha.

El comando DIR es pre-procesado en la función __Dir() durante el tiempo de compilación.

__Dir() es una función de compatibilidad, esta fue superada por DIRECTORY(), la cual devuelve toda la información en un arreglo multidimensional.

Examples

```
__Dir() // Información de todos los DBF en el directorio actual
__Dir( "*.dbf" ) // Lista todos los DBF en el directorio actual

// Lista todos los PRG de la librería de ejecución (RTL) de Harbour
// para sistemas operativos compatibles con DOS
__Dir( "c:\harbour\source\rtl\*.prg" )

// Lista todos los archivos de la sección pública sobre una máquina
// tipo Unix
__Dir( "/pub" )
```

Status

Ready

Compliance

Información de DBF: CA-Clipper muestra nombres de archivos en el formato 8.3, Harbour muestra los primeros 15 caracteres si un nombre largo de archivo esta disponible.

Listado de archivos: para formatear los nombres mostrados usamos algo así como: PADR(Nombre, 8) + " " + PADR(Ext, 3) CA-Clipper usa nombres de archivo 8.3, con Harbour probablemente se podría cortar los nombres largos de archivo para llenar este molde.

Files

La librería es rtl

See Also:

[ADIR\(\)](#)
[ARRAY\(\)](#)
[ARRAY\(\)](#)
[DIR](#)

DIR

Muestra el listado de archivos

Syntax

```
DIR [<cFileMask>]
```

Arguments

<cFileMask> Máscara de archivos para incluir en el retorno de la función. Esta podría contener subdirectorios (path) y caracteres estándar usados como comodines, según sean soportados por el sistema operativo (como * y ?). Si <cFileMask> no contiene la ruta al archivo entonces SET DEFAULT es usado para mostrar archivos en la máscara.

Description

Si ninguna <cFileMask> es dada, __Dir() muestra información acerca de todos los *.DBF en la ruta SET DEFAULT. esta información contiene: - Nombre del archivo - Número de registros - Fecha de la última actualización - Tamaño de cada archivo.

Si <cFileMask> es dada, __Dir() lista todos los archivos que coinciden con la máscara en los siguientes detalles: Nombre, Extensión, Tamaño, Fecha.

El comando DIR es pre-procesado en la función __Dir() durante el tiempo de compilación.

__Dir() es una función de compatibilidad, esta fue superada por DIRECTORY(), la cual devuelve toda la información en un arreglo multidimensional.

Examples

```
__Dir() // Información de todos los DBF en el directorio actual

__Dir( "*.dbf" ) // Lista todos los DBF en el directorio actual

// Lista todos los PRG de la librería de ejecución (RTL) de Harbour
// para sistemas operativos compatibles con DOS
__Dir( "c:\harbour\source\rtl\*.prg" )

// Lista todos los archivos de la sección pública sobre una máquina
// tipo Unix
__Dir( "/pub" )
```

Status

Ready

Compliance

Información de DBF: CA-Clipper muestra nombres de archivos en el formato 8.3, Harbour muestra los primeros 15 caracteres si un nombre largo de archivo está disponible.

Listado de archivos: para formatear los nombres mostrados usamos algo así como: PADR(Nombre, 8) + " " + PADR(Ext, 3) CA-Clipper usa nombres de archivo 8.3, con Harbour probablemente se podría cortar los nombres largos de archivo para llenar este molde.

See Also:

[ADIR\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__Dir\(\)*](#)

ADIR()

Llena arrays pre-definidos con información de archivo / directorio

Syntax

```
ADIR( [<cFileMask>], [<aNombre>], [<aTamano>], [<aFecha>],  
[<aHora>], [<aAtrib>] ) -> nEntradasDirectorio
```

Arguments

<cFileMask> Máscara de archivos para incluir en el retorno de la función. Esta podría contener subdirectorios (path) y caracteres estandar usados como comodines, segun sean soportados por el sistema operativo (como * y ?). Si <cFileMask> no contiene la ruta al archivo entonces SET DEFAULT es usado para mostrar archivos en la máscara.

<aNombre> Array para llenar con el Nombre de los archivos que cumplen con <cFileMask>. Cada elemento es una cadena de caracteres que incluye el Nombre y Extensión del archivo sin la ruta de acceso. Nombre es el nombre largo de archivo como es reportado por el sistema operativo y no necesariamente en el formato mayúsculas 8.3 del D.O.S.

<aTamano> Array para llenar con el Tamaño de los archivos que cumplen con <cFileMask>. Cada elemento es un número entero con el tamaño del archivo en bytes. Los Directorios siempre tienen un tamaño cero.

<aFecha> Array para llenar con la Fecha de la ultima modificación del archivo que cumplen con <cFileMask>. Cada elemento es del tipo "Date"

<aHora> Array para llenar con la Hora de la ultima modificación del archivo que cumplen con <cFileMask>. Cada elemento es una cadena de caracteres en el formato: HH:MM:SS.

<aAtrib> Array para llenar con los atributos de los archivos que cumplen con <cFileMask>. Cada elemento es una cadena de caracteres, Vea DIRECTORY() por información sobre los valores de los atributos. Si Ud. pasa un array a <aAtrib>, la función va a devolver archivos con los atributos Normal, Oculto (H), sistema (S) y directorio (D) Si <aAtrib> no es especificado o es distinto de un array solo archivos con atributo normal porian ser devueltos.

el comportamiento de esta función con los atributos que poseen las maquinas tipo Unix.

Returns

ADIR() retorna el número de entradas de archivo que cumplen con la condición establecida en la máscara <cFileMask>.

Description

ADIR() retorna el número de archivos y/o directorios que cumplen con un formato especificado, este tambien llena una serie de arrays con Nombre, Tamaño, Fecha, Hora y Atributo de estos archivos. El array pasado debe ser pre-inicializado al tamaño apropiado, vea el ejemplo más abajo. Con motivo de incluir los atributos Oculto (H), sistema (S) o de directorio (D) <aAtrib> debe ser especificado.

ADIR() es una función de compatibilidad, esta fue superada por DIRECTORY(), la cual devuelve toda la información en un arreglo multidimensional.

Examples

```
LOCAL aNombre, aTamano, aFecha, aHora, aAtrib, nLen, i  
nLen := ADIR( "*.JPG" ) // Nro de archivos JPG files en directorio  
  
IF nLen > 0  
  aNombre := Array( nLen ) // hace lugar para guardar la información  
  aTamano := Array( nLen )  
  aFecha := Array( nLen )  
  aHora := Array( nLen )  
  aAtrib := Array( nLen )  
  FOR i = 1 TO nLen  
    ? aNombre[i], aTamano[i], aFecha[i], aHora[i], aAtrib[i]  
  NEXT  
ELSE  
  ? "Este directorio no tiene ni pelusa"
```

ENDIF

Status

Ready

Compliance

<aNombre> esta yendo a ser llenado con nombres largos de archivo y no necesariamente con el formato mayúsculas 8.3 del D.O.S.

Files

La librería es rtl

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

DISKSPACE()

Obtiene la cantidad de espacio disponible en el disco

Syntax

```
DISKSPACE( [<nDisco>] [, <nTipo>] ) --> nDiskbytes
```

Arguments

<nDrive> es el número de disco del que esta solicitando información donde 1 = A, 2 = B, etc, Si se especifica cero ó ningún parametro DISKSPACE() trabaja sobre la unidad actual de disco Por defecto es cero.

<nTipo> es el tipo de espacio que está siendo requerido. Por defecto es HB_DISK_AVAIL.

Returns

<nDiskBytes> es el número de bytes en la unidad especificada que coincide con el tipo requerido.

Description

Por defecto esta función retorna el número de bytes de espacio libre en el disco actual que esta disponible para el usuario que solicita la información.

Hay 4 tipos de información disponible:

HB_FS_AVAIL La cantidad de espacio disponible para el usuario que hace la solicitud. Este valor podría ser menor que HB_FS_FREE si las asignaciones de espacio en disco (quotas) son soportadas por el sistema operativo al tiempo de ejecución y estas quotas están asignadas. De otro modo el valor será igual al retornado por HB_FS_FREE.

HB_FS_FREE La cantidad actual de espacio libre en el disco.

HB_FS_USED El número de bytes en uso en el en el disco.

HB_FS_TOTAL La cantidad total de espacio asignado para el usuario si las quotas estan asignadas. De otro modo el tamaño actual del disco.

Si la información es requerida sobre un disco que no esta disponible un error de ejecución 2018 será establecido.

Examples

```
? "Ud. tiene disponible: " + STR( DISKSPACE() ) + " bytes " +;  
  "sobre un total de " + STR( DISKSPACE( 0, HB_FS_TOTAL) )
```

Note: Ver ..\tests\tstdspac.prg por otros ejemplos.

Status

Started

Compliance

CA-Clipper retorna un valor entero cuya utilidad esta limitada a discos menores de 2 gigabytes. La versión de Harbour retorna un valor de punto flotante con 0 decimales si el disco es > 2 gigabytes. <nTipo> es una extensión de Harbour.

Platforms

Dos, Win32, OS/2, Unix

Files

El código fuente está en diskspac.c La librería asociada es rtl El archivo de cabecera es fileio.ch

ERRORSYS()

Instala el manejador de errores por defecto

Syntax

```
ERRORSYS() --> NIL
```

Arguments

Returns

ERRORSYS() siempre retorna NIL.

Description

ERRORSYS() es llamado en el inicio por Harbour e instala el manejador de errores por defecto. Normalmente no se debe llamar a esta función directamente. En su lugar use ERRORBLOCK() para instalar su propio manejador de errores.

Status

Ready

Compliance

Esta función es 100 % Clipper compatible

Files

El código fuente está en errorsys.c La librería asociada es rtl

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

EVAL()

Evalúa un bloque de código (codeblock)

Syntax

```
EVAL( <bBloque> [, <xVal> [,...]]) --> xExpresion
```

Arguments

- <bBloque> Bloque de código a ser evaluado.
- <xVal> Argumento para ser pasado al bloque de código.
- <xVal...> Lista de argumentos para ser pasados al bloque de código.

Returns

EVAL() retorna <xExpresion>, el valor de la última expresión dentro del bloque. El valor devuelto puede ser de cualquier tipo válido.

Description

Esta función evalúa el bloque de código expresado como <bBloque> al ejecutarlo y pasarle los parámetros como argumentos ,luego retorna su valor evaluado. Si hay múltiples expresiones dentro del bloque de código, la última expresión será el valor de esta función.

Si el bloque de código requiere parámetros para ser pasados a éste, ellos son especificados en la lista de parámetros <xVal>. Cada parámetro es separado por una coma dentro de la lista de parámetros y cada expresión es separada por comas dentro de la lista de expresiones.

Nota: Un bloque de código es un valor de datos especial que hace referencia a código de programa compilado (contiene código ejecutable), puede incluso contener funciones y el hecho que permite exportar variables estáticas y locales dentro de él, la da una gran versatilidad. Aunque las macros y los bloques de código son similares, las macros son cadenas de caracteres que se compilan durante la ejecución de un programa y se ejecutan a continuación. Los bloques de código se compilan junto con los otros PRG durante la compilación del programa. Por esto son más rápidos, Es posible compilar un bloque de código en tiempo de ejecución desde una cadena de caracteres usando el operador de macros (&). Un bloque de código esta compuesto por: { |<lista argumentos>| <lista expresiones> } Los separadores verticales deben estar presentes aunque no el bloque no reciba argumentos.

Examples

```
bBloque = {|arg1, arg2| QOUT( arg1+arg2) }  
EVAL( bBloque, "Harbour", " es fabuloso")
```

Tests

Ver ejemplos.

Status

Ready

Compliance

Esta función es 100 % compatible con CA-Clipper.

Platforms

Todas

Files

La librería es vm

See Also:

[AEVAL\(\)](#)

[ARRAY\(\)](#)

Recolector de memoria

Léame con las características de la recolección de memoria en Harbour.

Description

El recolector de memoria (garbage collector) usa la siguiente lógica: - primero recolectar todas las ubicaciones de memoria que puedan constituirse en "basura" - luego inspeccionar todas las variables, por si esos bloques están todavía referenciados.

Note que sólo arrays, objetos y bloques de código son recolectados porque esos son los únicos tipos de datos que pueden causar: auto-referencias (a[1]:=a) ó referencias circulares: (a[1]:=b; b[1]:=c; c[1]:=a) que no pueden ser apropiadamente desasignadas por un simple conteo de referencia.

Como todas las variables en Harbour son almacenadas dentro de algunas tablas disponibles (eval stack, tabla de memvars y array de variables estáticas), entonces chequear si la referencia es todavía activa es bastante fácil y no requiere ningún tratamiento especial durante la asignación de memoria. Adicionalmente el recolector de memoria inspecciona algunos datos internos usados por la implementación de objetos de Harbour que también almacena algunos valores que pueden contener referencias de memoria. Estos datos son usados para inicializar variables de instancia de la clase, y son almacenadas en variables compartidas por la clase.

En casos especiales cuando el valor de una variable de Harbour es almacenada internamente en algún área estática (a nivel de lenguaje C ó assembler), por ejemplo SETKEY() almacena bloques de código que serán evaluados cuando se presione una tecla, el recolector de memoria no será capaz de inspeccionar esos valores porque este no conoce su ubicación. Esto podría ocasionar que algunos bloques de memoria sean liberados prematuramente. Para prevenir la prematura desasignación de esos bloques ellos deben ser bloqueados para el recolector de memoria. Para ello se definen distintos estados del bloque de memoria: #define HB_GC_UNLOCKED 0 /* desbloqueado */ #define HB_GC_LOCKED 1 /* No recolectar el bloque de memoria */ #define HB_GC_USED_FLAG 2 /* bit para la bandera usado/sin uso */

El bloque de memoria puede ser bloqueado con hb_gcLockItem(), método recomendado si un ítem de estructura es usado ó la función hb_gcLock() un puntero directo a memoria es usado. El bloque de memoria puede ser desbloqueado por hb_gcUnlockItem() ó hb_gcUnlock().

Nótese sin embargo que todas las variables pasadas a una función de bajo nivel son pasadas mediante la pila de evaluación (eval stack), así ellas no necesitan bloquearse durante la llamada a la función. El bloque puede ser requerido, si un valor pasado es copiado dentro de algún área estática para hacerla disponible para otras funciones de bajo nivel, llamadas después de la salida de la función que almacena el valor. Esto es requerido porque el valor es removido de la pila de evaluación después de la llamada a la función y esta no puede seguir siendo referenciada por otras variables.

Sin embargo la inspección de todas las variables puede ser una operación de un gran consumo de tiempo. Esto requiere que todos los arrays asignados tengan que ser recorridos a través de todos sus elementos para encontrar más arrays. También todos los bloques de código son inspeccionados, en busca de variables locales separadas que ellos están referenciando. Por esta razón, la búsqueda por bloques de memoria no referenciados es realizada durante los estados inactivos.

El estado inactivo es el estado cuando no hay un código real de la aplicación ejecutándose. Por ejemplo, el código del usuario es detenido durante 0.1 segundo por INKEY(0.1) - Harbour esta chequeando sólo el teclado durante este tiempo. Esto deja sin embargo suficiente tiempo para muchas otras tareas en segundo plano. Una de esas tareas en segundo plano, puede ser la búsqueda de bloques de memoria no referenciados.

Asignando memoria

El recolector de memoria, recoge bloques de memoria asignados con llamadas a la función hb_gcAlloc(). La memoria asignada por hb_gcAlloc() debería ser liberada con la función hb_gcFree().

Bloqueando memoria

La memoria asignada con `hb_gcAlloc()` debería ser bloqueada para prevenir una automática liberación como un puntero de memoria si no es almacenado dentro de una variable a nivel de Harbour. Todos los valores de Harbour (items), almacenados internamente en áreas estáticas de lenguaje C deben ser bloqueadas. Vea `hb_gcLockItem()` y `hb_gcUnlockItem()` para más información.

La recolección de memoria

Durante la búsqueda de memoria no referenciada, el recolector de memoria (RM) está usando un algoritmo llamado "mark & sweep", marcar y barrer. Este es realizado en tres etapas:

- 1) Marcar todos los bloques asignados por el RM con un bandera: "sin uso"
- 2) barrer (buscar) todos los lugares conocidos y limpiar las banderas sin uso por los bloques de memoria que son referenciados allí;
- 3) finalizar recolectando por desasignación de todos los bloques de memoria que aún estan marcados como sin uso y que no están bloqueados.

Para acelerar las cosas un poco, la etapa de marca es simplificada por la inversión del significado de la bandera "sin uso". Después de la desasignación de los bloques sin uso, todos los bloques todavía activos son marcados con la bandera "usado" así nosotros podemos invertir el significado de esta bandera al estado "sin uso" en la próxima recolección. Todos los bloques de memoria nuevos ó sin bloquear son automáticamente marcados como "sin uso" usando la bandera actual, lo cual asegura que todos los bloques de memoria son marcados con la misma bandera antes de que la etapa de barrido comience.

Ver `hb_gcCollectAll()` y `hb_gcItemRef()`

Llamando al recolector de memoria desde código Harbour

El RM puede ser llamado directamente desde un programa en Harbour. Esto es útil en situaciones donde no hay estados inactivos disponibles ó la aplicación esta trabajando en un bucle sin interacción con el usuario y hay muchas asignaciones de memoria. Vea `HB_GCALL()` por una explicación de como llamar a esta función desde el código de Harbour.

See Also:

[hb_gcAlloc\(\)](#)
[hb_gcFree\(\)](#)
[hb_gcLockItem\(\)](#)
[hb_gcUnlockItem\(\)](#)
[hb_gcCollectAll\(\)](#)
[hb_gcItemRef\(\)](#)
[HB_GCALL\(\)](#)
[HB_IDLESTATE\(\)](#)

hb_gcAlloc()

Asigna memoria que será recolectada por el recolector de memoria.

Syntax

```
#include <hbapi.h>
void *hb_gcAlloc( ULONG ulSize, HB_GARBAGE_FUNC_PTR pCleanupFunc );
```

Arguments

<ulSize> es el tamaño solicitado del bloque de memoria.

<pCleanupFunc> es un Puntero a la función HB_GARBAGE_FUNC que será llamada directamente antes de la liberación del bloque de memoria sin uso ó NULL. Esta función debería liberar toda otra memoria asignada y almacenada dentro del bloque de memoria. Por ejemplo, esta libera todos los items almacenados dentro del array. La función recibe un sólo parámetro: el puntero a la memoria asignada por hb_gcAlloc().

Returns

irrecuperable.

Description

hb_gcAlloc() es usada para asignar la memoria que será rastreada por el RM. Este permite una apropiada liberación de memoria en el caso de variables auto-referenciadas ó con referencias cruzadas a nivel de Harbour. La memoria asignada con esta función debería ser liberada con la función hb_gcFree() ó esta será automáticamente desasignada por el RM si no esta bloqueada ó si no esta referenciada por alguna variable a nivel de Harbour.

Examples

Vea ../source/vm/arrays.c

```
PHB_BASEARRAY pArr = (PHB_BASEARRAY) hb_gcAlloc( sizeof( HB_BASEARRAY),
                                                hb_arrayReleaseGarbage );
```

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcFree\(\)](#)
[hb_gcLockItem\(\)](#)
[hb_gcUnlockItem\(\)](#)

hb_gcFree()

Libera la memoria que fué asignada con hb_gcAlloc().

Syntax

```
void hb_gcFree( void *pMemoryPtr );
```

Arguments

`<pMemoryPtr>` es el puntero a la memoria a liberar. Este puntero de memoria debe ser asignado con la función hb_gcAlloc().

Returns

Description

La función hb_gcFree() es usada para liberar la memoria que fué asignada con la función hb_gcAlloc().

Examples

```
Vea ../source/vm/arrays.c  
hb_gcFree( (void *) pBaseArray ); // puntero al array a liberar
```

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcAlloc\(\)](#)

[hb_gcLockItem\(\)](#)

[hb_gcUnlockItem\(\)](#)

hb_gcLockItem()

Bloquea la memoria para prevenir la desasignación por el RM.

Syntax

```
void hb_gcLockItem( HB_ITEM_PTR pItem );
```

Arguments

<pItem> es el puntero a la estructura item que será bloqueada. El item pasado puede ser de cualquier tipo de datos, aunque arrays objetos y bloques de códigos son bloqueados solamente. Otros tipos de datos no necesitan bloqueo así que ellos son simplemente ignorados.

Returns

Description

La función hb_gcLockItem() es usada para bloquear el puntero de memoria almacenado en la estructura item pasada. Este suprime la liberación de memoria si el RM no encuentra alguna referencia a este puntero. El RM almacena un contador de bloqueo y cada llamada a esta función incrementa el contador. El item es bloqueado si el contador es mayor que cero.

Examples

```
Vea ../source/rtl/setkey.c
// bloquea un codeblock para prevenir la liberación por el RM
hb_gcLockItem( sk_list_tmp-> pAction );
```

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcAlloc\(\)](#)

[hb_gcFree\(\)](#)

[hb_gcUnlockItem\(\)](#)

hb_gcUnlockItem()

Desbloquea la memoria para prevenir la liberación por el RM

Syntax

```
void hb_gcUnlockItem( HB_ITEM_PTR pItem );
```

Arguments

<pItem> es el puntero a la estructura item que será bloqueada. El item pasado puede ser de cualquier tipo de datos, aunque arrays objetos y bloques de códigos son bloqueados solamente. Otros tipos de datos no necesitan bloqueo así que ellos son simplemente ignorados.

Returns

Description

La función hb_gcUnlockItem() es usada para desbloquear el puntero de memoria almacenado en la estructura item pasada, que fué previamente bloqueada con una llamada a hb_gcLockItem(). Esto permite liberar la memoria durante la recolección de memoria sin uso si el RM no encuentra ninguna referencia a este puntero. El RM almacena el contador de bloqueo, cada llamada a esta función decrementa el contador. Esta función no libera la memoria almacenada dentro del item, la memoria debe ser desasignada sin embargo durante la recolección de memoria sin uso más cercana si el contador de bloqueo es igual a cero y el puntero de memoria no es referenciado por ninguna variable a nivel de Harbour.

Examples

```
Vea ../source/rtl/setkey.c  
hb_gcUnlockItem( sk_list_tmp-> pAction ); // libera el item
```

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcAlloc\(\)](#)

[hb_gcFree\(\)](#)

[hb_gcLockItem\(\)](#)

hb_gcCollectAll()

Examina todos los bloques de memoria y libera la memoria sin uso.

Syntax

```
void hb_gcCollectAll( void );
```

Arguments

Returns

Description

Esta función examina la pila de evaluación, las tablas de memvars, el array de variables estáticas y las tablas de clases creadas en busca de bloques de memoria referenciados. Después de examinar todos los bloques de memoria sin uso y los bloques que no están bloqueados, son liberados.

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcAlloc\(\)](#)

[hb_gcFree\(\)](#)

[hb_gcLockItem\(\)](#)

[hb_gcUnlockItem\(\)](#)

hb_gcItemRef()

Marca la memoria para prevenir la desasignación por el RM.

Syntax

```
void hb_gcItemRef( HB_ITEM_PTR pItem );
```

Arguments

<pItem> es el puntero a la estructura item que será examinada. El item pasado puede ser de cualquier tipo de datos, aunque arrays objetos y bloques de códigos son bloqueados solamente. Otros tipos de datos no necesitan bloqueo así que ellos son simplemente ignorados.

Returns

Description

El recolector de memoria usa la función hb_gcItemRef() durante la inspección de punteros de memoria referenciados. Esta función chequea el tipo del item pasado y examina recursivamente todos los otros bloques de memoria referenciados por este item, si éste es un array un objeto ó un bloque de código

NOTA: Esta función es reservada para el recolector de memoria (RM) solamente. Esta NO debe ser llamada desde el código del usuario llamarla puede causar resultados impredecibles (bloques de memoria referenciados por el item pasado pueden ser liberados prematuramente durante la recolección de memoria más cercana).

Status

Clipper

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcAlloc\(\)](#)

[hb_gcFree\(\)](#)

[hb_gcLockItem\(\)](#)

[hb_gcUnlockItem\(\)](#)

HB_GCALL()

Inspecciona la memoria y libera todos los bloques de memoria sin uso.

Syntax

```
HB_GCALL()
```

Arguments

Returns

Description

Esta función libera todos los bloques de memoria que son considerados como "basura".

Status

Harbour

Compliance

Esta función es una extensión de Harbour.

Platforms

Todas

Files

Archivo fuente: ../source/vm/garbage.c

See Also:

[hb_gcCollectAll\(\)](#)

hb_setInitialize ()

Syntax

Prototipo C

```
#include <hbset.h>
hb_setInitialize( void ) --> void
```

Arguments

Returns

Description

Status

Ready

El cumplimiento no es aplicable a las llamadas de API.

Files

La biblioteca es rtl

Platforms

Todas

hb_setRelease()

Syntax

Prototipo C

```
#include <hbset.h>
hb_setRelease( void ) --> void
```

Arguments

Returns

Description

Status

Ready

El cumplimiento no es aplicable a las llamadas de API.

Files

La biblioteca es rtl

Platforms

Todas

HB_LANGSELECT()

Selecciona un módulo de mensajes específico a una nacionalidad.

Syntax

```
HB_LANGSELECT(<cNuevoLeng>) --> cViejoLeng
```

Arguments

<cNuevoLeng> El código de identificación del módulo de idioma del país. Los valores posibles para <cNuevoLeng> están mostrados abajo, tal como están definidos en la librería de Lenguajes, ordenados por idioma.

Returns

<cViejoLeng> El anterior identificador de Lenguaje

Description

Esta función establece un módulo de lenguaje o idioma para las advertencias internas, mensajes NatMsg y errores internos que utilizan nombres de fechas, días, meses, etc. Cuando una identificación de Lenguaje es elegida todos los mensajes son mostrados de acuerdo al actual idioma, hasta que otro sea seleccionado, ó el programa termine. La ID son dos letras que establecen la correspondencia con un idioma dado de acuerdo a una tabla.

Nota: La tabla que se muestra arriba puede no estar completa.

Examples

- * El siguiente ejemplo cambia el módulo de idioma por defecto, y luego muestra por pantalla la ID del módulo de lenguaje, el nombre del día de la semana y el mes en varios idiomas.

```
REQUEST HB_LANG_PT
REQUEST HB_LANG_RO
REQUEST HB_LANG_ES
LOCAL nViejo
```

```
HB_LANGSELECT("PT") // el idioma por defecto es ahora Portugués
? "La nueva ID de idioma elegida es ", HB_LANGSELECT() // PT
? CDOW( DATE() )
? CMONTH( DATE() )
```

```
nViejo := HB_LANGSELECT("RO") // el idioma por defecto es Romano
? "La vieja ID de idioma elegida era ", nViejo // PT
? "La nueva ID de idioma elegida es ", HB_LANGSELECT() // RO
? CDOW( DATE() )
? CMONTH( DATE() )
```

```
HB_LANGSELECT("ES") // el idioma por defecto es ahora Español
? "La nueva ID de idioma elegida es ", HB_LANGSELECT() // ES
? CDOW( DATE() )
? CMONTH( DATE() )
```

Tests

Vea ..\tests\langapi.prg

Status

Ready

Compliance

Esta función es una Extensión de Harbour.

Platforms

Dos, Win32, OS/2

Files

La librería asociada es rtl

See Also:

[HB_LANGNAME\(\)](#)

[ARRAY\(\)](#)

HB_LANGNAME()

Retorna el nombre del módulo actual de lenguaje en uso.

Syntax

```
HB_LANGNAME() --> cNombreLenguaje
```

Arguments

Returns

HB_LANGNAME() retorna <cNombreLenguaje>, el nombre del idioma en uso.

Description

Esta función describe el nombre del idioma ó lenguaje correspondiente al que existe por defecto ó ha sido establecido por HB_LANGSELECT().

Examples

- * El siguiente ejemplo cambia el módulo de idioma por defecto, y luego muestra por pantalla la ID del módulo de lenguaje, el idioma asociado a esa ID. y el día de la semana y el mes en varios idiomas.

```
REQUEST HB_LANG_PT  
REQUEST HB_LANG_ES
```

```
HB_LANGSELECT("PT") // el idioma por defecto es ahora Portugués  
? "El idioma actual es ", HB_LANGNAME() // Portugués  
? "La nueva ID de idioma elegida es ", HB_LANGSELECT() // RO  
? CDOW( DATE() )  
? CMONTH( DATE() )
```

```
HB_LANGSELECT("ES") // el idioma por defecto es ahora Español  
? "El idioma actual es ", HB_LANGNAME() // Español  
? CDOW( DATE() )  
? CMONTH( DATE() )
```

Tests

Vea ..\tests\langapi.prg

Status

Ready

Compliance

Esta función es una Extensión de Harbour.

Platforms

Dos, Win32, OS/2

Files

La librería asociada es lang Nota: el nombre de la extensión de la librería, puede cambiar con el sistema operativo

See Also:

[HB_LANGSELECT\(\)](#)
[ARRAY\(\)](#)

MEMOTRAN()

Reemplaza los retornos del carro/nueva linea de una cadena

Syntax

```
MEMOTRAN( <cCadena>, <cDuro>, <cBlando> ) --> <cCadenaConvertida>
```

Arguments

<cCadena> es la cadena de caracteres a convertir.

<cDuro> es el carácter para reemplazar los retornos de carro "duro". Si no es especificado por defecto es el punto y coma.

<cBlando> es el carácter para reemplazar los retornos de carro "blandos" Si no es especificado, por defecto es un espacio en blanco.

Returns

MEMOTRAN() retorna <cCadenaConvertida>, la cadena transformada.

Description

Esta función retorna una cadena donde los caracteres de retorno de carro han sido convertidos a los caracteres especificados.

Examples

- * El siguiente ejemplo formatea un campo memo conteniendo un mensaje de error en una cadena adecuada para ser enviada a la función ALERT()

```
cMensaje = MEMOTRAN( Errores->MENSAJE )  
ALERT( cMensaje, aOpciones )
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

La librería asociada es rtl

See Also:

[HARDCR\(\)](#)

[ARRAY\(\)](#)

HARDCR ()

Reemplaza los CHR(141) por retornos de carro normal CHR(13)

Syntax

```
HARDCR( <cCadena> ) --> <cCadenaConvertida>
```

Arguments

<cCadena> es la cadena de caracteres a convertir.

Returns

HARDCR() retorna <cCadenaConvertida>, la cadena transformada.

Description

Esta función retorna una cadena/memo donde los caracteres de retorno de carro automaticos ó "blandos" (CHR(141)) son convertidos a caracteres de retorno de carro forzado ó "duro" CHR(13).

Examples

* El siguiente ejemplo asigna a una variable de cadena, el campo memo NOTAS existente en la base Clientes, transformado para mostrarlo por pantalla.

```
cNotas = HARDCR( Clientes->NOTAS )  
? cNotas
```

* El siguiente ejemplo envia a la impresora el contenido de un campo memo.

```
SET DEVICE TO PRINTER  
DEVPOS( nFil, nCol)  
DEVOUT( HARDCR( Clientes->NOTAS ) )
```

Status

Ready

Compliance

Esta función es totalmente compatible con CA-Clipper.

Files

La librería asociada es rtl

See Also:

[MEMOTRAN \(\)](#)

[ARRAY \(\)](#)

SETMODE()

Cambia el modo de video a un número especificado de filas y columnas

Syntax

```
SETMODE( <nFil>, <nCol> ) --> lSatisfactorio
```

Arguments

<nFil> es el número de filas para el modo de video deseado.

<nCol> es el número de columnas para el modo de video deseado.

Returns

SETMODE() devuelve verdadero si el cambio en el modo de video fue satisfactorio, de otra manera devuelve falso.

Description

SETMODE() es una función que cambia el modo de video (dependiendo de la combinación de tarjeta de video y monitor), para que coincida con el número de filas y columnas especificado. Nótese que sólo hay realmente unas pocas combinaciones de pares de filas/columnas que produzcan el cambio en el modo de video. Los siguientes están disponibles para D.O.S: 12 filas x 40 columnas
12 filas x 80 columnas 25 filas x 40 columnas 25 filas x 80 columnas
28 filas x 40 columnas 28 filas x 80 columnas 50 filas x 40 columnas
43 filas x 80 columnas 50 filas x 80 columnas Algunas modos sólo están disponibles para monitor color y/o VGA. Cualquier cambio producido en el tamaño de la pantalla es actualizado en los valores devueltos por MAXROW() y MAXCOL().

p El primer ejemplo cambia al modo de visualización de 12 líneas:

```
IF SETMODE( 12, 40)
? "Oye hombre, eres tu chicato ?"
ELSE
? "Mam tr eme los anteojos !"
ENDIF
```

p El próximo ejemplo cambia al modo de visualización de 50 líneas:

```
IF SETMODE( 50, 80)
? "Este maravilloso modo de video ha sido seteado"
ELSE
? "Oye Manuel que este monitor no est hecho de goma !"
ENDIF
```

Status

Ready

Compliance

Algunos de estos modos no están disponibles en Clipper

Platforms

DOS

Files

El Archivo fuente es gtdos.c

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

ABS()

Devuelve el valor absoluto de un número.

Syntax

```
ABS(<nNumero>) --> <nAbsNumero>
```

Arguments

<nNumero> Cualquier número.

Returns

<nAbsNumero> El valor absoluto de un número.

Description

Esta función entrega el valor absoluto de un valor o una expresión <nNumero>.

Examples

```
Function Main()  
  
    Local nNumero := 50  
    Local nNumero1 := 27  
  
    qout( nNumero - nNumero1 )  
    qout( nNumero1 - nNumero )  
    qout( ABS(nNumero - nNumero1 ) )  
    qout( ABSnNumero1 - nNumero ) )  
    qout( ABS( -1 * 345 ) )  
Return Nil
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[EXP\(\)](#)

EXP()

Calcula el valor de e elevado a la potencia indicada.

Syntax

EXP(<nNumero>) --> <nValor>

Arguments

<nNumero> Cualquier número real.

Returns

<nValor> El anti-logaritmo de <nNumero>.

Description

Esta function devuelve el valor de e elevado a la potencia indicada en <nNumero>. Es la contraria de LOG().

Examples

? EXP(45)

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[LOG\(\)](#)

INT()

Devuelve la parte entera de un valor numérico.

Syntax

```
INT( <nNumero> ) --> <nIntNumero>
```

Arguments

<nNumero> Cualquier valor numérico.

Returns

<nIntNumero> La parte entera de un valor numérico.

Description

Esta función convierte una expresión numérica en entero. Todos los dígitos decimales son truncados. Esta función no redondea el valor ni por encima ni por debajo; simplemente trunca el valor desde el punto decimal.

Examples

```
SET Decimal to 5
? INT(632512.62541)
? INT(845414111.91440)
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[ROUND\(\)](#)

[ARRAY\(\)](#)

LOG()

Devuelve el logaritmo natural de un número.

Syntax

LOG(<nNumero>) --> <nLog>

Arguments

<nNumero> Cualquier expresión numérica.

Returns

<nExponente> El logaritmo natural de <nNumero>.

Description

Esta función devuelve el logaritmo natural del número <nNumero>. Si <nNumero> es 0 o menor que 0, se produce un desbordamiento numérico, el cual es mostrado en la pantalla como una serie de asteriscos. Esta función es la contraria de EXP().

Examples

? LOG(632512)

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[EXP\(\)](#)

MAX()

Devuelve el mayor de dos números o fechas.

Syntax

```
MAX(<xValor>,<xValor1>) --> <xMax>
```

Arguments

<xValor> Cualquier fecha o valor numérico.

<xValor1> Cualquier fecha o valor numérico (con el mismo tipo de <xValor>).

Returns

<xMax> El número mayor (o la fecha mayor).

Description

Esta función devuelve el mayor de las dos expresiones pasadas. Si <xValor> y <xValor1> son del tipo numérico, el valor devuelto por esta función será también numérico y será el mayor de los números pasados a dicha función. Si <xValor> y <xValor1> son del tipo fecha el valor devuelto será también de tipo fecha. Este será la mayor de ellas.

Examples

```
? MAX(214514214,6251242142)
? MAX(CTOD('11/11/2000'),CTOD('21/06/2014'))
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[MIN\(\)](#)

MIN()

Determina el menor de dos números o fechas.

Syntax

```
MIN(<xValor>,<xValor1>) --> <xMin>
```

Arguments

<xValor> Cualquier fecha o valor numérico.

<xValor1> Cualquier fecha o valor numérico (con el mismo tipo de <xValor>).

Returns

<xMin> El valor o fecha menor.

Description

Esta función devuelve el menor valor del par de expresiones recibidas. <xValor> y <xValor1> deben ser del mismo tipo. Si es numérico, el menor valor es devuelto. Si es fecha, la fecha menor es devuelta.

Examples

```
? MIN(214514214,6251242142)  
? MIN(CTOD('11/11/2000'),CTOD('21/06/2014'))
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[MAX\(\)](#)

MOD()

Devuelve el residuo de dos números.

Syntax

```
MOD( <nNumero>,<nNumero1> ) --> <nResiduo>
```

Arguments

<nNumero> Numerador en una expresión de división.

<nNumero1> Denominador en una expresión de división.

Returns

<nResiduo> El residuo despues de la operación de división.

Description

Esta función devuleve el residuo de un número dividido entre otro.

Examples

```
? MOD(12,8.521)
? Mod(12,0)
? Mod(62412.5142,4522114.12014)
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[ARRAY\(\)](#)

SQRT()

Calcula la raíz cuadrada de un número.

Syntax

```
SQRT( <nNumero> ) --> <nSqrt>
```

Arguments

<nNumero> Cualquier valor numérico.

Returns

<nSqrt> La raíz cuadrada de <nNumero>.

Description

Esta función devuelve la raíz cuadrada de <nNumero>. La precisión de esta evaluación esta basada únicamente en la asignación del comando SET DECIMAL TO. Cualquier número negativo pasado como <nNumero> siempre retornará 0.

Examples

```
SET Decimal to 5
? SQRT(632512.62541)
? SQRT(845414111.91440)
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[ROUND\(\)](#)

ROUND ()

Redondea una expresión numérica.

Syntax

```
ROUND( <nNumero>, <nLugares> ) --> <nResultado>
```

Arguments

<nNumero> Cualquier valor numérico.

<nLugares> El número de lugares decimales a redondear.

Returns

<nResultado> El número redondeado.

Description

Esta función redondea el valor de <nNumero> a la cantidad de lugares decimales especificados en <nLugares>. Si el valor de <nLugares> es un número negativo, la función intentará redondear <nNumero> en el número completo. Números de 5 a 9 serán redondeados hacia arriba, el resto redondeados abajo.

Examples

```
? ROUND(632512.62541,5)
? ROUND(845414111.91440,3)
```

Status

Ready

Compliance

Esta es una función compatible con CA-Clipper.

Platforms

Todas

Files

Se encuentra en la librería rtl

See Also:

[INT\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

hb_getMathError()

Obtener el último error de la librería matemática.

Syntax

C Prototype

```
#include <hbmath.h>
hb_getMathError (void) --> int iMathError
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_resetMathError()

Inicializa el error matemático.

Syntax

C Prototype

```
#include <hbmath.h>
hb_resetMathError (void) --> void
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_isMathHandler()

Verifica si el manejador de errores de Harbour esta disponible.

Syntax

C Prototype

```
#include <hbmath.h>
hb_isMathHandler (void) --> int iIsMathHandler
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_installMathHandler()

Agrega un manejador de errores definido por el usuario la cadena de manejadores de error.

Syntax

C Prototype

```
#include <hbmath.h>
hb_installMathHandler (HB_MATH_HANDLERPROC handlerproc) --> HB_MATH_HANDLERHANDLE
handle
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_deinstallMathHandler()

Quita un manejador de errores definido por el usuario de la cadena de manejadores de error.

Syntax

C Prototype

```
#include <hbmath.h>
hb_deinstallMathHandler (HB_MATH_HANDLERHANDLE handle) --> int iSuccess
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_setMathHandlerStatus()

Pone el estado del manejador de errores definido por el usuario en la cadena de manejadores de

Syntax

C Prototype

```
#include <hbmath.h>
hb_setMathHandlerStatus (HB_MATH_HANDLERHANDLE handle, int status) --> int iSuccess
```

Arguments

nuevo valor del estado, puede ser uno de los siguientes:

HB_MATH_HANDLER_STATUS_INACTIVE --> indentificador presente pero no activo

HB_MATH_HANDLER_STATUS_ACTIVE --> indentificador presente y activo

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

hb_getMathHandlerStatus()

Lee el estado del manejador de errores definido por el usuario en la cadena de manejadores de e

Syntax

C Prototype

```
#include <hbmath.h>
hb_getMathHandlerStatus (HB_MATH_HANDLERHANDLE handle) --> int iStatus
```

Arguments

Returns

Description

Status

Ready

Compatibilidad no es aplicable a las llamadas a las APIs.

Files

Se encuentra en la librería rtl

Platforms

Todas

License

Licencia Actual Del Proyecto Harbour

Description

LICENCIA

Traducción: Alejandro de Gárate. <alex_degarate@hotmail.com>

Nota del traductor: Se encaró la traducción de la licencia de Harbour de su original en inglés al español, debido a que su comprensión podría aportar beneficios en los alcances del uso de Harbour y evitar algunos malentendidos ya sea para uso personal ó comercial.

Soy conciente que la presente traducción puede contener algún error u omisión, aunque he hecho los esfuerzos posibles para que ello no ocurra.

Para mayor información sobre que opina la FSF acerca de las traducciones de la GPL consulte: <<http://gnu.org/copyleft/copyleft.es.html>>

Nota Previa al texto de la licencia: (requisito de la FSF)

This is an unofficial translation of the GNU General Public License into spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL--only the original English text of the GNU GPL does that. However, we hope that this translation will help spanish speakers understand the GNU GPL better.

Esta es una traducción no oficial de la Licencia Publica General GNU al español. No fue publicada por la Fundación de Software Libre, y no establece legalmente los términos de distribución para software que usa la GPL GNU--sólo el texto original en inglés de la GPL GNU lo hace. Sin embargo, esperamos que esta traducción ayude a gente de habla hispana a entender mejor la GNU GPL.

LICENCIA ACTUAL DEL PROYECTO HARBOUR

=====

(Actualizada al 10/06/2001)

"Este programa es software libre; Ud. puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General GNU como fué publicada por la Fundación de Software Libre; sea la versión 2 de la Licencia, ó (a su elección) alguna versión posterior.

Este programa es distribuido con la esperanza que le pueda ser útil, pero SIN NINGUNA GARANTIA; aún sin la garantía implícita de COMERCIALIZACION ó DESTINO PARA ALGÚN PROPÓSITO PARTICULAR. Consulte la License Pública General GNU para mayores detalles.

Ud. debería haber recibido una copia de la Licencia Pública General GNU junto con este programa; Vea el archivo COPYING. Si no, escriba a la Fundación de Software Libre:

Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA (ó visite su sitio Web en <http://www.gnu.org/>).

Como una especial excepción, el proyecto Harbour da permiso para usos adicionales del texto contenido en esta liberación de Harbour.

La excepción es que, si Ud. enlaza (link) la Librerías de Harbour con otros archivos para producir un ejecutable, esto por sí mismo no causa que el ejecutable resultante sea cubierto por la Licencia Pública General GNU. El uso de ese ejecutable por parte suya no esta en ninguna forma restringido, en cuenta del enlazado del código de librería de Harbour dentro de él.

Esta excepción sin embargo, no invalida algunas otras razones por las cuales el archivo ejecutable podría estar cubierto por la Licencia Pública General GNU.

Esta excepción se aplica: solamente al código liberado por el Proyecto Harbour bajo el nombre "Harbour". Si Ud. copia código desde otro Proyecto Harbour ó desde alguna liberación de la Fundación de Software Libre, dentro de una copia de

Harbour, tal como la Licencia Pública General lo permite, la excepción no se aplica al código que Ud. agrega de esta forma. Para evitar engaños a alguien con el estado de estos archivos modificados, Ud. debe borrar el aviso de excepción de ellos.

Si Ud. escribe modificaciones de su propia creación para Harbour, es su elección, cuándo permitir que esta excepción se aplique a sus modificaciones. Si Ud. no desea eso, borre el aviso de excepción."

fin del archivo es_licence.txt

See Also:

[OVERVIEW](#)

Estado de Espera

Archivo Léame para Estado de Espera

Description

El estado de espera ó inactivo es el estado de la máquina virtual de Harbour cuando esta espera por una acción del usuario a través del teclado ó el mouse. El estado inactivo ocurre actualmente durante las llamadas a INKEY(). Todas las funciones que no usen la función INKEY() pueden señalar el estado inactivo con una llamada a la función Harbour HB_IDLESTATE() ó hb_idleState() a nivel de lenguaje C.

Durante los estado de espera, la máquina virtual llama al recolector de memoria y este puede llamar a acciones definidas por el usuario (tareas en segundo plano ó background). Este también libera porciones del tiempo de la CPU para algunas plataformas deficientes que no son lo suficientemente inteligentes (Windows NT).

Por definición de tareas en segundo plano vea las funciones: HB_IDLEADD() y HB_IDLEDEL().

Para llamar directamente a acciones en segundo plano vea la función: HB_IDLESTATE().

Para indicar el estado inactivo desde código en lenguaje C vea la hb_idleState() en la API,

See Also:

[HB_IDLEADD\(\)](#)

[HB_IDLEDEL\(\)](#)

HB_IDLEADD()

Agrega tareas en segundo plano.

Syntax

```
HB_IDLEADD( <cbAccion> ) --> nHandle
```

Arguments

<cbAccion> es un bloque de código que será ejecutado durante los estados de espera, No hay argumentos pasados a este bloque de código durante la evaluación.

Returns

<nHandle> El manejador (un valor entero) que identifica la tarea. Este manejador puede ser usado para eliminar la tarea.

Description

HB_IDLEADD() agrega el pasado bloque de código a las lista de tareas en segundo plano que serán evaluadas durante los estados inactivos. No hay un límite para el número de tareas.

Examples

```
nTarea := HB_IDLEADD( {|| SayTime()} )
```

Status

Ready

Compliance

Extensión de Harbour, similar a la función FT_ONIDLE() disponible en la librería Nanforum.

Platforms

Todas

Files

El archivo fuente es ../source/rtl/idle.c

See Also:

[HB_IDLEDEL\(\)](#)

[HB_IDLESTATE\(\)](#)

HB_IDLEDEL()

Remueve tareas desde la lista de tareas en segundo plano.

Syntax

```
HB_IDLEDEL( <nHandle> ) --> xAccion
```

Arguments

<nHandle> es el identificador de la tarea devuelto por la función HB_IDLEADD().

Returns

Description

La función HB_IDLEDEL() remueve la acción asociada con el identificador pasado desde la lista de tareas en segundo plano. El identificador debería ser el valor retornado por la llamada previa a la función HB_IDLEADD(). Si la tarea especificada está definida, entonces el codeblock es retornado, de otro modo el valor NIL es retornado.

Examples

```
nTarea := HB_IDLEADD( {|| SayTime()} )
INKEY(10)
cbAccion := HB_IDLEDEL( nTarea )
```

Status

Ready

Compliance

Extensión de Harbour

Platforms

Todas

Files

El archivo fuente es ../source/rtl/idle.c

See Also:

[HB_IDLEADD\(\)](#)

[HB_IDLESTATE\(\)](#)

HB_IDLESTATE()

Evalúa una sola tarea en segundo plano y llama al recolector

Syntax

```
HB_IDLESTATE()
```

Arguments

Returns

Description

La función HB_IDLESTATE() solicita la recolección de memoria sin uso y ejecuta una única tarea definida por el bloque de código pasado con la función HB_IDLEADD(). Cada llamada a esta función evalúa una tarea diferente en el orden de la creación de tareas. No hay argumentos pasados durante la evaluación del bloque de código. Esta función puede ser llamada con seguridad, aún si no hay tareas definidas en segundo plano.

Examples

```
nTarea1 := HB_IDLEADD( { || MuestreLaHora() } )
nTarea2 := HB_IDLEADD( { || SaveScreen() } )

DO WHILE( ! bTerminado )
    bTerminado := HagaAlgoMuyImportante()
    HB_IdleState()
ENDDO

cbAccion := HB_IDLEDEL( nTarea1 )
HB_IDLEDEL( nTarea2 )
```

Status

Ready

Compliance

Extensión de Harbour, similar a la función FT_IAMIDLE() disponible en la librería Nanforum.

Platforms

Todas

Files

El archivo fuente es ../source/rtl/idle.c

See Also:

[HB_IDLEADD\(\)](#)

[HB_IDLEDEL\(\)](#)

hb_idleState()

Evalúa una sola tarea en segundo plano y llama al recolector

Syntax

```
void hb_idleState( void );
```

Arguments

Returns

Description

La función `hb_idleState()` es una función en lenguaje C que solicita la recolección de memoria sin uso y ejecuta una sola tarea en segundo plano, definida por el bloque de código pasado con la función `HB_IDLEADD()`. Esta también libera porciones de tiempo de CPU para aquellas plataformas que lo requieran.

Cada llamada a esta función evalúa una tarea diferente en el orden de la creación de tareas. No hay argumentos pasados durante la evaluación del bloque de código.

Esta función puede ser llamada con seguridad, aún si no hay tareas definidas en segundo plano.

Esta función es automáticamente llamada desde la función `INKEY()`.

Status

Ready

Platforms

Todas

Files

El archivo fuente es `../source/rtl/idle.c`

See Also:

[HB_IDLEADD\(\)](#)

[HB_IDLEDEL\(\)](#)

[HB_IDLESTATE\(\)](#)

__dbCopyStruct()

Crea otra base de datos basada en la estructura actual del DBF

Syntax

```
__dbCopyStruct( <cNombreArch>, [<aListaCampos>] ) --> NIL
```

Arguments

<cNombreArch> es el nombre de una nueva base de datos a crear. (.dbf) es la extensión por defecto si ninguna es especificada.

<aListaCampos> es un array donde cada elemento es un nombre de campo. Los nombres pueden ser especificados en mayúsculas ó minúsculas.

Returns

__dbCopyStruct() siempre retorna NIL.

Description

La función **__dbCopyStruct()** crea una nueva base de datos vacía, con una estructura que esta basada en la base de datos actualmente abierta en este área de trabajo. Si **<aListaCampos>** esta vacía, el archivo recientemente creado podría tener la misma estructura que la base de datos actualmente abierta. De otro modo el nuevo archivo podría contener sólo campos que coincidan exactamente con **<aListaCampos>**.

__dbCopyStruct() puede ser usada para crear un subconjunto de la base de datos actualmente abierta, basada en una lista dada de campos.

El comando COPY STRUCTURE es pre-procesado en la función **__dbCopyStruct()** durante el tiempo de compilación.

Examples

```
// Crea un nuevo archivo que contiene la misma estructura.
USE TEST
__dbCopyStruct( "MyCopy.DBF" )

// Crea un nuevo archivo que contiene parte de la estructura
// original
LOCAL aLista
USE TEST
aLista := { "NOMBRE" }
__dbCopyStruct( "nombre.DBF", aLista )
```

Status

Ready

Compliance

__dbCopyStruct() trabaja exactamente como la función de CA-Clipper, **__dbCopyStruct()**.

Platforms

Todas

Files

La librería es rdd

See Also:

[COPY STRUCTURE](#)

[COPY STRUCTURE EXTENDED](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__dbCopyXStruct\(\)](#)

[__dbCreate\(\)](#)

[__dbStructFilter\(\)](#)

COPY STRUCTURE

Crea otra base de datos basada en la estructura actual del DBF

Syntax

```
COPY STRUCTURE TO <xcNombreArchivo> [FIELDS <campo,...>]
```

Arguments

TO <xcNombreArchivo> es el nombre del nuevo archivo de base de datos a crear. (.dbf) es la extensión por defecto si ninguna es dada.

expresión de caracteres encerrada entre paréntesis.

FIELDS <campo,...> es una lista opcional de nombres de campos a copiar desde la base de datos actualmente abierta en el orden especificado, por defecto son todos los campos. Los nombres pueden ser especificados en mayúsculas ó minúsculas.

Description

COPY STRUCTURE crea una nueva base de datos vacía con una estructura que esta basada en la base de datos actualmente abierta en este área de trabajo.

COPY STRUCTURE puede ser usada para crear un subconjunto de la base de datos actualmente abierta, basada en una lista dada de campos.

El comando COPY STRUCTURE es pre-procesado en la función `__dbCopyStruct()` durante el tiempo de compilación.

Examples

```
// Crea un nuevo archivo que contiene la misma estructura.
USE TEST
COPY STRUCTURE TO MiCopia
```

```
// Crea un nuevo archivo que tiene parte de la estructura original
USE TEST
COPY STRUCTURE TO UnaParte FIELDS nombre, direccion
```

Status

Ready

Compliance

COPY STRUCTURE trabaja exactamente como en CA-Clipper

Platforms

Todas

See Also:

[COPY STRUCTURE EXTENDED](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__dbCopyStruct\(\)](#)

[__dbCopyXStruct\(\)](#)

[__dbCreate\(\)](#)

[__dbStructFilter\(\)](#)

__dbCopyXStruct()

Copia la estructura actual de la base a un archivo de definición.

Syntax

```
__dbCopyXStruct( <cNombreArchivo> ) --> lExito
```

Arguments

<cNombreArchivo> es el nombre del archivo de definición de destino a crear, (*.dbf) es la extensión por defecto si ninguna es dada.

Returns

__dbCopyXStruct() retorna (.F.) si ninguna base de datos está en uso en el área actual de trabajo, (.T.) si el proceso fué exitoso, ó un error en tiempo de ejecución si la operación de creación del archivo ha fallado.

Description

La función __dbCopyXStruct() crea una nueva base de datos llamada <cNombreArchivo> con una estructura predefinida (también llamado "archivo de estructura extendida"):

Cada registro en el nuevo archivo contiene información acerca de un campo en el archivo original. CREATE FROM podría ser usado para crear la base de datos desde el archivo de estructura extendida.

Por razones prehistóricas de compatibilidad, los campos de caracteres mayores de 255 caracteres son tratados en una forma especial al escribir parte de la longitud en el campo FIELD_DEC de acuerdo a la siguiente fórmula (Esto es hecho internamente):

```
FIELD->FIELD_DEC := int( nLength / 256 )  
FIELD->FIELD_LEN := ( nLength % 256 )
```

Luego si se desea calcular la longitud de un campo, se puede usar la siguiente fórmula:

```
nLength := IIF( FIELD->FIELD_TYPE == "C", ;  
               FIELD->FIELD_DEC * 256 + FIELD->FIELD_LEN, ;  
               FIELD->FIELD_LEN )
```

El comando COPY STRUCTURE EXTENDED es pre-procesado en la función __dbCopyXStruct() durante el tiempo de compilación.

Examples

```
// Abre la base de datos, y copia su estructura a un nuevo archivo.  
USE Test  
__dbCopyXStruct( "TestEstr" )  
  
// Abre el nuevo archivo y lista todos sus registros  
USE TestEstr  
LIST
```

Status

Ready

Compliance

__dbCopyXStruct() trabaja exactamente como la función de CA-Clipper, __dbCopyXStruct().

Platforms

Todas

Files

La librería es rdd

See Also:

[COPY STRUCTURE](#)

[COPY STRUCTURE EXTENDED](#)

[CREATE](#)

[CREATE FROM](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__dbCopyStruct\(\)](#)

[__dbCreate\(\)](#)

COPY STRUCTURE EXTENDED

Copia la estructura actual de la base a un archivo de definición.

Syntax

```
COPY STRUCTURE EXTENDED TO <xcNombreArchivo>
```

Arguments

TO <xcNombreArchivo> es el nombre del nuevo archivo de base de datos a crear, (.dbf) es la extensión por defecto si ninguna es dada. Esta puede ser especificada como un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

Description

COPY STRUCTURE EXTENDED crea una nueva base de datos llamada <cNombreArchivo> con una estructura predefinida (también llamada "archivo de estructura extendida"):

Cada registro en el nuevo archivo contiene información acerca de un campo en el archivo original. CREATE FROM podría ser usado para crear la base de datos desde el "archivo de estructura extendida".

Por razones prehistóricas de compatibilidad, los campos de caracteres mayores de 255 caracteres son tratados en una forma especial al escribir parte de la longitud en el campo FIELD_DEC de acuerdo a la siguiente fórmula (Esto es hecho internamente):

```
<fixed>
FIELD->FIELD_DEC := int( nLength / 256 )
FIELD->FIELD_LEN := ( nLength % 256 )
```

Luego si se desea calcular la longitud de un campo, se puede usar la siguiente fórmula:

```
nLength := IIF( FIELD->FIELD_TYPE == "C", ;
                FIELD->FIELD_DEC * 256 + FIELD->FIELD_LEN, ;
                FIELD->FIELD_LEN )
```

El comando COPY STRUCTURE EXTENDED es pre-procesado en la función __dbCopyXStruct() durante el tiempo de compilación.

```
// Abre la base de datos, y copia su estructura a un nuevo archivo.
USE Test
__dbCopyXStruct( "TestEstr" )

// Abre el nuevo archivo y lista todos sus registros
USE TestEstr
LIST
```

Status

Ready

Compliance

COPY STRUCTURE EXTENDED trabaja exactamente como en CA-Clipper.

Platforms

Todas

See Also:

[COPY STRUCTURE](#)
[CREATE](#)

CREATE FROM

ARRAY()

ARRAY()

__dbCopyStruct()

__dbCopyXStruct()

__dbCreate()

__dbCreate()

Crea una estructura extendida ó usa una para crear otro archivo

Syntax

```
__dbCreate( <cNombreArchivo>, [<cDesdeArchivo>], [<cNombreRDD>],  
[<lNuevo>], [<cAlias>] ) --> lUsado
```

Arguments

<cNombreArchivo> es el nombre del archivo destino a crear y luego abrir, (.dbf) es la extensión por defecto si ninguna es especificada.

<cDesdeArchivo> es el nombre de una estructura extendida opcional desde la cual el archivo destino <cNombreArchivo> va a ser construido. Si es omitida, un nuevo archivo de estructura extendida vacía con el nombre <cNombreArchivo> es creado y abierto, en el área de trabajo actual.

<cNombreRDD> es el nombre del RDD para crear el archivo destino, Si es omitido, es usado el RDD por defecto.

<lNuevo> es una expresión lógica opcional, (.T.) abre el nombre del archivo de destino <cNombreArchivo> en la próxima área disponible sin uso y hace de esta el área actual. (.F.) abre el archivo de destino en el área de trabajo actual. El valor por defecto es (.F.). El valor de <lNuevo> es ignorado si <cDesdeArchivo> no es especificado.

<cAlias> es un alias opcional para usar el archivo de destino con el comando USE. Si no es especificado, alias esta basado en el nombre <cNombreArchivo>.

Returns

__dbCreate() retorna (.T.) si hay una base de datos usada en el área actual de trabajo (esta podría ser la nueva area de trabajo elegida), ó (.F.) si no hay ninguna base de datos usada. Note que si hubo éxito debería ser devuelto (.F.), pero si hay alguna falla probablemente se termine con un error en tiempo de ejecución y no con un valor de (.F.)

Description

La función __dbCreate() funciona en dos modos dependiendo del valor de <cDesdeArchivo>:

1) Si **<cDesdeArchivo>** está vacío o no está especificado, un nuevo archivo de estructura extendida vacía, con el nombre <cNombreArchivo> es creado y luego abierto, en el área actual de trabajo (<lNuevo> es ignorado). El nuevo archivo tiene la siguiente estructura:

El comando CREATE es pre-procesado en la función __dbCopyStruct() durante el tiempo de compilación y usa este modo.

2) Si **<cDesdeArchivo>** es especificado, este es abierto y se asume como un archivo de estructura extendida donde cada registro contiene al menos los siguientes campos (en cualquier orden): FIELD_NAME, FIELD_TYPE, FIELD_LEN y FIELD_DEC. Cualquier otro campo es ignorado. Con esta información el archivo <cNombreArchivo> es creado y abierto en el nuevo area de trabajo (de acuerdo a <lNuevo>) si esta es una nueva área de trabajo, esta se vuelve la actual.

Por razones prehistóricas de compatibilidad, los campos de caracteres del archivo de estructura extendida los cuales sean mayores de 255 caracteres deberían ser tratados en una forma especial al escribir parte de la longitud en el campo FIELD_DEC de acuerdo a la siguiente fórmula:

```
FIELD->FIELD_DEC := int( nLength / 256 )  
FIELD->FIELD_LEN := ( nLength % 256 )
```

El comando CREATE FROM es pre-procesado en la función __dbCopyStruct() durante el tiempo de compilación.

```
// Crea un nuevo archivo de estructura extendida, agrega algunos  
// registros y luego crea un nueva base de datos desde este
```

```
// archivo (CREATE FROM)

__dbCreate( "molde" )
DBAPPEND()
FIELD->FIELD_NAME := "CANAL"
FIELD->FIELD_TYPE := "N"
FIELD->FIELD_LEN := 2
FIELD->FIELD_DEC := 0
DBAPPEND()
FIELD->FIELD_NAME := "PROGRAMA"
FIELD->FIELD_TYPE := "C"
FIELD->FIELD_LEN := 20
FIELD->FIELD_DEC := 0
DBAPPEND()
FIELD->FIELD_NAME := "RESUMEN"
FIELD->FIELD_TYPE := "C" // este campo es de 1000 char longitud
FIELD->FIELD_LEN := 232 // 1000 % 256 = 232
FIELD->FIELD_DEC := 3 // 1000 / 256 = 3
DBCLOSEAREA()
__dbCreate( "tv_guia", "molde" )
```

Status

Ready

Compliance

__dbCreate() trabaja exactamente como en CA-Clipper.

Platforms

Todas

Files

La librería es rdd

See Also:

[COPY STRUCTURE](#)

[COPY STRUCTURE EXTENDED](#)

[CREATE](#)

[CREATE FROM](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__dbCopyStruct\(\)](#)

[__dbCopyXStruct\(\)](#)

CREATE

Crea un archivo de estructura extendida vacío.

Syntax

```
CREATE <xcNombreArchivo> [VIA <xcNombreRDD>] [ALIAS <xcAlias>]
```

Arguments

<xcNombreArchivo> es el nombre del archivo de destino a crear y abrir, (*.dbf) es la extensión por defecto si ninguna es dada. Este puede ser especificada un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

VIA <xcNombreRDD> es el nombre del RDD con el cual se va a crear el archivo de destino. Si es omitido, el RDD por defecto es usado. Este puede ser especificado como un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

<cAlias> es un alias opcional para usar el archivo de destino con el comando USE. Si no es especificado, alias esta basado en el nombre <cNombreArchivo>.

Description

El comando CREATE, crea un nuevo archivo de estructura extendida vacío <cNombreArch> y luego lo abre en el área actual de trabajo. El nuevo archivo tiene la siguiente estructura:

El comando CREATE es pre-procesado en la función __dbCopyStruct() durante el tiempo de compilación y usa este modo.

Examples

```
// Crea un nuevo archivo de estructura extendida, agrega algunos
// registros y luego crea un nueva base de datos desde este archivo
// (CREATE FROM)

CREATE molde
APPEND BLANK
FIELD->FIELD_NAME := "CANAL"
FIELD->FIELD_TYPE := "N"
FIELD->FIELD_LEN := 2
FIELD->FIELD_DEC := 0
APPEND BLANK
FIELD->FIELD_NAME := "PROGRAMA"
FIELD->FIELD_TYPE := "C"
FIELD->FIELD_LEN := 20
FIELD->FIELD_DEC := 0
APPEND BLANK
FIELD->FIELD_NAME := "RESUMEN"
FIELD->FIELD_TYPE := "C" // este campo es de 1000 char longitud
FIELD->FIELD_LEN := 232 // 1000 % 256 = 232
FIELD->FIELD_DEC := 3 // 1000 / 256 = 3
CLOSE
CREATE tv_Guia FROM molde
```

Status

Ready

Compliance

CREATE trabaja exactamente como en CA-Clipper.

Platforms

Todas

See Also:

[COPY STRUCTURE](#)

[COPY STRUCTURE EXTENDED](#)

[CREATE FROM](#)

ARRAY()

ARRAY()

_dbCopyStruct()

_dbCopyXStruct()

_dbCreate()

CREATE FROM

Crea una base de datos desde un archivo de estructura extendida

Syntax

```
CREATE <xcNombreArchivo> FROM <xcDesdeArchivo> [VIA <xcNombreRDD>]  
[NEW] [ALIAS <xcAlias>]
```

Arguments

<xcNombreArchivo> es el nombre del archivo de destino a crear y abrir, (*.dbf) es la extensión por defecto si ninguna es dada. Este puede ser especificada un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

FROM <xcDesdeArchivo> es el nombre del archivo de estructura extendida desde la cual el archivo de destino va a ser construido. Este puede ser especificado como un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

VIA <xcNombreRDD> es el nombre del RDD con el cual se va a crear el archivo de destino. Si es omitido, el RDD por defecto es usado. Este puede ser especificado como un literal de nombre de archivo ó como una expresión de caracteres encerrada entre paréntesis.

NEW abre el nombre del archivo de destino <cNombreArch> en la próxima área disponible sin uso y hace de ésta el área actual. Si es omitido, abre el archivo de destino en el área de trabajo actual.

ALIAS <xcAlias> es el alias opcional del archivo de destino para usar con el comando USE. Si ninguno es especificado el alias esta basado en el nombre <cNombreArchivo>.

Description

El comando CREATE FROM abre un archivo de estructura extendida <xcDesdeArchivo> donde cada registro contiene al menos los siguientes campos (en cualquier orden): FIELD_NAME, FIELD_TYPE, FIELD_LEN y FIELD_DEC. Cualquier otro campo es ignorado. Con esta información el archivo <cNombreArchivo> es creado y abierto en el nuevo area de trabajo (de acuerdo a la cláusula NEW) si ésta es una nueva área de trabajo, esta se vuelve la actual.

Por razones prehistóricas de compatibilidad, los campos de caracteres del archivo de estructura extendida los cuales sean mayores de 255 caracteres deberían ser tratados en una forma especial al escribir parte de la longitud en el campo FIELD_DEC de acuerdo a la siguiente fórmula:

```
FIELD->FIELD_DEC := int( nTamano / 256 )  
FIELD->FIELD_LEN := ( nTamano % 256 )
```

El comando CREATE FROM es preprocesado en la función __dbCopyStruct() durante el tiempo de compilación y usa este modo.

Examples

Vea el ejemplo del comando CREATE.

Status

Ready

Compliance

CREATE FROM trabaja exactamente como en CA-Clipper.

Platforms

Todas

See Also:

[COPY STRUCTURE](#)

[COPY STRUCTURE EXTENDED](#)

[CREATE](#)

ARRAY()

ARRAY()

_dbCopyStruct()

_dbCopyXStruct()

_dbCreate()

__FLEDIT()*

Filtra un array con estructura de base de datos.

Syntax

```
__FLEDIT( <aEstructura>, [<aListaCampos>] ) --> aEstructuraFiltrada
```

Arguments

<aEstructura> es un array multidimensional con la estructura de los campos de la base de datos, la cual es usualmente la salida de DBSTRUCT(), donde cada elemento del array tiene la siguiente estructura:

<aListaCampos> es un array donde cada elemento es un nombre de campo. Los nombres pueden ser especificados en mayúsculas o en minúsculas.

Returns

__FLEDIT() retorna un nuevo array multidimensional donde cada elemento esta en la misma estructura que el original <aEstructura>, pero el array es construido de acuerdo a la lista de campos en <aListaCampos>. Si <aListaCampos> esta vacío, **__FLEDIT()** retorna una referencia al array original <aEstructura>.

Description

__FLEDIT() puede ser usado para crear un subconjunto de la estructura de la base de datos, basada en una lista de campos dadas.

Note que los nombres de campos en <aEstructura> DEBEN ser especificados en mayúsculas de lo contrario podría no encontrarse coincidencia.

SET EXACT no tiene efecto en el valor retornado.

__FLEDIT() es una función de compatibilidad y es un sinónimo para **__dbStructFilter()** la cual hace exactamente lo mismo.

Examples

```
LOCAL aEstructura, aLista, aRet
aEstructura := { { "CODIGO", "N", 4, 0 }, ;
                { "NOMBRE", "C", 10, 0 }, ;
                { "TELEF", "C", 13, 0 }, ;
                { "IQ", "N", 3, 0 } }
aLista := { "IQ", "NOMBRE" }
aRet := __FLEDIT( aEstructura, aLista )
        // { { "IQ", "N", 3, 0 }, { "NAME", "C", 10, 0 } }

aRet := __FLEDIT( aEstructura, {} )
? aRet == aEstructura // .T.

aLista := { "iq", "NOTEXIST" }
aRet := __FLEDIT( aEstruct, aLista )
        // { { "IQ", "N", 3, 0 } }

aLista := { "NOTEXIST" }
aRet := __FLEDIT( aEstruct, aLista ) // {}

// Crea un nuevo archivo que contiene parte de la estructura
// original
LOCAL aEstructura, aLista, aRet
USE TEST
aEstructura := DBSTRUCT()
aLista := { "NOMBRE" }
DBCREATE( "elnombre.dbf", __FLEDIT( aEstructura, aLista ) )
```

Status

Ready

Compliance

CA-Clipper tiene una función interna no documentada llamada **__FLEDIT()**, en

Harbour nosotros la llamamos `__dbStructFilter()`. El nuevo nombre da una mejor descripción de qué es lo que la función hace. En Harbour `__FLEDIT()`, simplemente llama a `__dbStructFilter()` y por tanto la segunda es la función recomendada.

Esta función es sólo visible si `../source/rdd/dbstrux.prg` fué compilado con la bandera `HB_C52_UNDOC`.

Platforms

Todas

Files

El archivo de cabecera es `dbstruct.ch` La librería es `rdd`

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[__dbCopyStruct\(\)](#)

[__dbStructFilter\(\)](#)

__dbStructFilter()

Filtra un array con estructura de base de datos.

Syntax

```
__dbStructFilter( <aEstructura>, [<aListaCampos>] )  
--> aEstructuraFiltrada
```

Arguments

<aEstructura> es un array multidimensional con la estructura de los campos de la base de datos, la cual es usualmente la salida de DBSTRUCT(), donde cada elemento del array tiene la siguiente estructura:

<aListaCampos> es un array donde cada elemento es un nombre de campo. Los nombres pueden ser especificados en mayúsculas o en minúsculas.

Returns

__dbStructFilter() retorna un nuevo array multidimensional donde cada elemento esta en la misma estructura que el original <aEstructura>, pero el array es construido de acuerdo a la lista de campos en el array <aListaCampos>. Si <aListaCampos> esta vacío, __dbStructFilter() retorna una referencia al array original <aEstructura>.

Description

__dbStructFilter() puede ser usado para crear un subconjunto de la estructura de la base de datos, basada en una lista de campos dadas.

Note que los nombres de campos en <aEstructura> DEBEN ser especificados en mayúsculas de lo contrario podría no encontrarse coincidencia.

SET EXACT no tiene efecto en el valor retornado.

Examples

```
LOCAL aEstructura, aLista, aRet  
aEstructura := { { "CODIGO", "N", 4, 0 }, ;  
                { "NOMBRE", "C", 10, 0 }, ;  
                { "TELEF", "C", 13, 0 }, ;  
                { "IQ", "N", 3, 0 } }  
aLista := { "IQ", "NOMBRE" }  
aRet := __dbStructFilter( aEstructura, aLista )  
        // { { "IQ", "N", 3, 0 }, { "NAME", "C", 10, 0 } }  
  
aRet := __dbStructFilter( aEstructura, {} )  
? aRet == aEstruct // .T.  
  
aLista := { "iq", "NOTEXIST" }  
aRet := __dbStructFilter( aEstructura, aLista )  
        // { { "IQ", "N", 3, 0 } }  
  
aLista := { "NOTEXIST" }  
aRet := __dbStructFilter( aEstruct, aLista ) // {}  
  
// Crea un nuevo archivo que contiene parte de la estructura  
// original  
LOCAL aEstruct, aLista, aRet  
USE TEST  
aEstruct := DBSTRUCT()  
aLista := { "NOMBRE" }  
DBCREATE( "elnombre.dbf", __dbStructFilter( aEstruct, aLista ) )
```

Status

Ready

Compliance

__dbStructFilter() es una extensión de Harbour. CA-Clipper tiene una función interna no documentada llamada __FLEDIT(), que hace lo mismo. El nuevo nombre da

una mejor descripción de que es lo que la función hace.

Platforms

Todas

Files

El archivo de cabecera es dbstruct.ch La librería es rdd

See Also:

[ARRAY\(\)](#)

[ARRAY\(\)](#)

[_dbCopyStruct\(\)](#)

[FLEEDIT\(\)*](#)

Extensiones a Harbour

Extensiones a Harbour

Description

Extensiones al lenguaje:

* Generación y manejo de Clases.

Clipper solamente permite la creación de objetos desde unas pocas clases estándar.

En Harbour, Ud. puede crear sus propias Clases completas con Métodos, Variables de Instancia, Variables de Clase y Herencia. Aplicaciones enteras pueden ser designadas y codificadas en un enfoque Orientado a Objetos.

* @<FunctionName>()

Retorna el puntero (la dirección) a la función.

El valor retornado no es útil a nivel de la programación de la aplicación pero es usado a bajo nivel para implementar el código Orientado a Objeto (Internamente, el método de una Clase es una función estática, no hay un símbolo para ella, entonces esta es accedida mediante su dirección).

* Clase TGetList

Soporte Orientado a Objetos para el manejo de GetList.

* ProcName() soporte para el nombre de Métodos de las Clases.

Los Métodos de Clases pueden ser recuperados de la pila de llamada.

* Memory() tiene nuevos valores de retorno.

Vea hbmemory.ch

* TRANSFORM() --> Agregado de nueva función para el formateo de cadenas:

@0 Hace el pad con ceros una cadena numérica.

* STOD() --> dDate

Nueva función que convierte una cadena en el formato yyyyymmdd a valor de fecha (date).

* Declaración STRONG TYPE Opcional en tiempo de compilación (y advertencias en tiempo de compilación de TYPE MISMATCH)

Example: LOCAL/STATIC Variable AS ...

* El depurador de Harbour provee de nuevas clases interesantes:

- La Clase TDbWindow podría ser la fundación de una ventana de visualización multiplataforma.

- La Clase TForm

- La Clase TDbMenu implementa ambos menús desplegables y (pulldown & popup)

Funcionalidad mejorada de la RTL:

- DIRECTORY(<cMascara>, <cBandera>, <lOchoPuntoTres>)

El tercer parámetro es un parámetro de Harbour (opcional) e indica que en aquellas plataformas que soportan nombres largos de archivo, Ud. solicita lo que podría considerarse el equivalente al nombre 8.3. Podría afectar ADIR() y DIR si ellas fueron modificadas para tomar ventaja de esto. Esta retornará nombres largos si el sistema operativo lo soporta.

- HB_DISKSPACE(<nDisco>, <nTipo>)

El segundo parámetro es un parámetro de Harbour (opcional) e indica el tipo de información del disco que esta siendo requerida. vea en ../doc/es/diskspac.txt para mayor información.

Utilidad Linea de Comandos

Opciones del Compilador

Description

Esta especificación sirve para CLIPPERCMD, HARBOURCMD, el compilador Harbour y las directivas #pragma en el código fuente.

La línea de comando siempre predomina sobre las variables de entorno.

Note que algunos switches no son aceptados en variables de entorno, y algunos otros en #pragmas.

Primero el analizador (parser) debería comenzar saltando a través de todos los tokens en la cadena separadas por espacios en blanco. (ó solo ir a través de todos los argv[])

1.) Si el token comienza con "-", este debe ser tratado como un nuevo estilo de switch .

Uno ó más caracteres switch pueden seguir a éste. El signo "-" dentro del token apaga el switch.

Si el switch tiene un argumento, todos los caracteres siguientes son tratados como parte del argumento:

El signo "/" n tiene significado especial aqui.

Switch	Opción resultante
-wn	(W N)
-w-n	(!W N)
-wi/harbour/include/	(W I=/harbour/include/)
-wi/harbour/include/n	(W I=/harbour/include/n)
-wes0n	(W ES=0 N)
-wen	(W [invalid switch: e] N)
-wesn	(W ES=default(0) N)
-wses	(W S ES=default(0))
-wess	(W ES=default(0) S)
-	([invalid switch])
-w-n-p	(!W !N P)
-w-n-p-	(!W !N !P)
-w- -w -w-	(finally: !W)

2.) Si el token comienza con "/", este debe ser tratado como un switch de compatibilidad.

El analizador revisa el token por el próximo signo "/", o EOS (End Of String = fin de línea) y trata la cadena resultante como un sólo switch.

Esto significa que si un switch con un argumento conteniendo el signo "/" tiene algunas limitaciones. Esto puede ser resuelto permitiendo el uso de caracteres comillas. Esto es mayormente un problema en sistemas que usan "/" como un separador de path.

El signo "-" n tiene significado especial aquí, éste no puede ser usado para deshabilitar un switch.

Switch	Opción resultante
/w/n	(W N)
/wo/n	([invalid switch: wo] N)

3.) Si el token comienza con otra cosa, éste debería ser saltado.

Los switches de Harbour son siempre insensibles a las mayúsculas.

En la línea de comandos de Harbour los dos estilos pueden ser usados conjuntamente:

```
HARBOUR -wnes2 /gc0/q0 -ic:\hello
```

Excepciones:

- El manejo del switch indocumentado /CREDIT en la línea de comandos de Harbour es inusual, revise el código actual por esto.

- El análisis de los parámetros en la línea de comandos de CLIPPER, HARBOUR y aplicaciones de Harbour, son cosas distintas, vea cmdarg.c por una nota.

Notas:

- En toda ocurrencia donde el path es aceptado, Harbour debería manejar el caracter doble comillas (") para especificar un path conteniendo espacios, signo negativo, barras u otros caracteres con un significado especial.

```
/i"c:/hello/"  
-i"c:/hello-n"  
/i"c:/program files/"  
-i"c:/program files/"
```

Sólo algunos ejemplos para las varias formas aceptadas:

```
//F20 == /F20 == F20 == F:20 == F20X  
//TMPPATH:C:\HELLO  
F20//TMPPATH:/TEMP///F:30000000 NOIDLE  
F0NOIDLEX10  
SQUAWKNOIDLE
```

"/" siempre deberían ser usados en la línea de comandos.

See Also:

[Opciones del Compilador](#)

Description

El Proyecto Harbour

```
*****  
* Este archivo contiene información sobre como obtener, instalar y *  
* usar Harbour. Por favor léalo *completamente* antes de solicitar *  
* ayuda *  
*****
```

Harbour es una implementación libre (en todo sentido) de un compilador de lenguaje xBase. Este está designado para ser compatible con el código fuente del compilador CA-Clipper(c). Esto significa que si Ud. tiene algún código que debería compilar usando CA-Clipper(c) entonces éste debería compilar también bajo Harbour.

La página web del Proyecto Harbour es:
<http://www.Harbour-Project.org/>

El estado y cualquier otra información está siempre disponible desde el sitio web. Hay una lista de correo de Harbour. Harbour todavía está en una etapa temprana de desarrollo, así que la lista de correo de los desarrolladores es prácticamente la única lista, aunque cualquiera es bienvenido para unirse a las discusiones.

Nos gustaría que Ud. se uniera al equipo de desarrollo de Harbour. Si Ud. está interesado, puede suscribirse a nuestra lista de correo y comenzar contribuyendo a este proyecto público libre.

Por favor sientáse con libertad de hacer preguntas, reportar ideas, sugerencias, correcciones, código, etc. Con la ayuda de todos Uds., el compilador Harbour y las librerías de ejecución se hará realidad muy pronto.

Qué es lo que esta distribución contiene ?

=====

Esta distribución es solamente una distribución de código fuente. Esta no contiene ningún archivo ejecutable. Las versiones ejecutables de Harbur estan dispobibles desde el sitio web. Las versiones ejecutables de Harbour NO crean programas ejecutables Harbour hasta este momento produce código de lenguaje C de salida, el cual debe ser compilado con la VM (Máquina Virtual de Harbour) y las librerías de soporte para crear programas funcionales. Por favor haga tests, corriendo Harbour contra su código fuente en Clipper y reportando cualquier problema que pudiera ocurrir.

Muy importante: El pre-procesador esta ahora trabajando.

Instalación

1. Deszipee el archivo zip de Harbour usando pkunzip ó equivalente.
Por ej.: pkunzip -d build37.zip
Esto creará el directorio Harbour y todos los subdirectorios relevantes.

2. Compile Harbour usando su compilador de lenguaje C. Archivos make para diferentes plataformas están incluidos.

--- COPYRIGHT ---

Qué información de copyright tenemos ?

--- LICENCIA ---

Información sobre la Licencia de uso de Harbour está disponible en el archivo LICENCE.TXT (cuando nosotros tengamos un licencia)

--- RENUNCIA ---

Los participantes del Proyecto Harbour no asumen responsabilidad por errores u omisiones en estos materiales.

ESTOS MATERIALES SON PROVISTOS "COMO ESTAN" SIN NINGUNA GARANTIA DE NINGUN TIPO, SEA EXPRESA O IMPLICITA, INCLUYENDO PERO NO LIMITADA A, LAS GARANTIAS IMPLICITAS DE COMERCIALIZACION, O EL USO PARA UN PROPOSITO PARTICULAR.

Los participantes del Proyecto Harbour no garantizan la exactitud ó comletitud del código, información, texto, salida ó cualquier otro item contenido dentro de esos materiales. Los participantes del Proyecto Harbour no serán responsables por ningun daño especial, directo, indirecto, incidental, ó consecuente, incluyendo pero no limitado a la pérdida de ingresos ó ganancias, los cuales pueden resultar del uso ó mal uso de estos materiales.

La información en el Proyecto Harbour esta sujeta a cambios sin previo aviso y no representa ningún compromiso futuro para los participantes del Proyecto Harbour.

El Proyecto Harbour

See Also:

[ARRAY\(\)](#)

Fuertemente tipeado

Chequeo de Tipo en Tiempo de Compilación

Description

El Strong Type Checking, podría ser también descripto como "Chequeo de tipo en Tiempo de Compilación". Como Ud. podría saber Clipper genera un error en Tiempo de Ejecución ("Type Mismatch") cuando nosotros intentamos realizar algunas operaciones con el tipo equivocado de variable.

Ejemplos:

```
LOCAL Var1 := "A"
? Var1 * 3          // Error
@ Var1, 7 SAY 'Hello' // Error
? SubStr( "Hello", Var1 ) // Error
```

Las 3 líneas superiores podrían todas resultar en un error de Tiempo de Ejecución, porque Var1 es del tipo CHARACTER, pero las líneas de arriba la usan como si fuera del tipo NUMERIC.

Usando un chequeo de Tipo Fuerte, ó Chequeo de Tipo en Tiempo de Compilación, el problema de arriba podría haber sido descubierto y reportado en tiempo de compilación, antes que esperar que el inevitable problema sea descubierto cuando finalmente nosotros ejecutemos el programa.

Los lenguajes fuertemente tipeados permiten al programador "decirle" al compilador (declarar) cual es el tipo de cada variable, así el compilador puede advertirle al programador, cuando esa variable declarada (Strong Typed), es usada en un contexto el cual es incompatible con su tipo declarado.

Por ejemplo si nosotros le "contamos" al compilador que la Var1 de arriba es del tipo CHARACTER (LOCAL Var1 AS CHARACTER), el compilador de Harbour, podría en devolución advertirnos que nosotros intentamos realizar el calculo:

```
Var1 * 3
```

porque el compilador sabe que nosotros no podemos realizar la mutiplicación de un CHARACTER. (Nosotros podríamos hacerlo en algún contexto pero eso esta mas allá del alcance de esta discusión). Similarmente nosotros podríamos haber sido advertidos cuando intentamos usar Var1 como un numero de Fila (@ Var1), ó como el segundo operando de la función SUBSTR(), SUBSTR("Hello", Var1) porque el compilador sabe que esas operaciones requieren un tipo NUMERIC antes que CHARACTER.

Lo de arriba puede ahorrarnos un montón de tiempo, señalandonos un problema del cual no podemos escapar, porque ese código nunca funcionará correctamente una vez ejecutado. Así que antes de esperar al ciclo de testeo, para que tales problemas sean descubiertos, (y algunas veces aún después de que nosotros hemos distribuido nuestras aplicaciones) en su lugar nosotros podemos saber de esos problemas tan pronto como nosotros tipeamos:

```
HARBOUR ProgName -w3
```

Harbour también ofrece un modo híbrido, donde puede reportar este tipo de problemas, aún sin el requerimiento que el programador declare el tipo de variables. Esta característica es referida como Chequeo Adaptativo de Tipo (Adaptive Type Checking). No es necesario que el programador haga ningún cambio en el código para tomar ventaja de ésta característica. Todos los 3 errores de arriba podrían haber sido reportados tan efectivamente como si el programador hubiera declarado Var1 como de Tipo Fuerte.

Harbour podría haber sido capaz de reportar tales problemas en tiempo de compilación, porque la asignación Var1 := "A" implica que Var1 es del tipo CHARACTER, hasta que a ésta le sea asignado otro valor. Así entonces Harbour podría "recordar" que Var1 a "adaptado" el tipo CHARACTER, y así la multiplicación subsecuente Var1 * 3 será reportada como un error, tan pronto como Ud. intente compilar tal código.

La parte hermosa de este modo híbrido, es que a diferencia de las variables Strong Typed, Ud. no tiene que declarar el código, así que no son necesarios cambios en el código fuente, el Tipo en su lugar es asumido por implicación (el Tipo del valor asignado). El otro beneficio es que, es completamente correcto

asignar un nuevo valor de diferente tipo, en cualquier momento, a esa variable no declarada (variante). Tan pronto como nosotros asignemos un nuevo tipo, el compilador entonces nos protegerá de usar la variable en un contexto incompatible, dado que la variable a "adaptado" este Tipo tan pronto como nosotros le hemos asignado un valor, el cual implica un Tipo.

Mientras el Chequeo Adaptativo de Tipo puede ser completamente efectivo en reportar muchos errores comunes, para tener todos los beneficios del Chequeo en Tiempo de Compilación es recomendado declarar el Tipo de las variables, cuando sea posible.

Las características del Tipo fuerte (Strong Type), también permiten la declaración de parámetros esperados (incluyendo opcionales) de las funciones definidas por el usuario, así como también el tipo retornado. Similarmente, Ud. puede declarar el tipo de cualquier variable de Clase, Métodos y Parámetros de Métodos.

__dbDelim()

Copia el contenido de una base de datos a un archivo de texto

Syntax

```
__dbDelim( <lExporta>, <xcArchivo>, [<xcDelim>], [<aCampos>],  
[<bFor>], [<bWhile>], [<nNext>], [<nRegistro>], <lResto> ) --> NIL
```

Arguments

<lExporta> es .F. agrega registros DESDE un archivo delimitado.

<xcArchivo> es el nombre del archivo de texto del cual se va a copiar ó al que se va agregar datos. Si no es especificada una extensión. ".txt" es usada por defecto.

<xcDelim> Cualquiera de los caracteres a usar como delimitadores de campo (sólo el primer caracter es usado). ó "BLANK" (en mayúsculas ó minúsculas), el cual elimina los caracteres delimitadores de campo y establece el separador de campo a un solo espacio en lugar de una coma.

<aCampos> Un array con los nombres de los campos a los que se limita el proceso. Si no es especificado, ó si está vacío, entonces todos los campos son procesados.

<bFor> Un bloque de código opcional conteniendo una expresión FOR que reducirá el número de registros a ser procesados.

<bWhile> Un bloque de código opcional conteniendo una expresión WHILE que reducirá el número de registros a ser procesados.

<nNext> Si está presente, pero <nRegistro> no está presente, especifica procesar este número de registros, empezando con el registro actual. Un valor de cero significa no procesar registros.

<nRegistro> Si está presente, especifica el único registro a procesar un valor de cero significa no procesar ningún registro. Predomina sobre <nNext> y <lResto>.

<lResto> Si <lExporta> es .T., y es establecido en .T. y no hay argumentos <nRegistro>, <nNext> ó <bWhile>, entonces procesa todos los registros, desde el actual hasta el último.

Returns

Description

__dbDelim() copia todo ó el contenido seleccionado, desde una tabla de base de datos a un archivo de texto SDF ó, agrega todo ó el contenido seleccionado desde un archivo de texto SDF a una tabla de base de datos.

Examples

```
// Copia cuentas delinquentes dentro de un archivo de texto  
// delimitado  
USE cuentas NEW  
COPY TO deudas DELIMITED FOR !EMPTY( cuentas->vencidas ) ;  
.AND. DATE() - cuentas->vencidas > 30  
  
// Importa nuevos registros de clientes  
USE clientes NEW  
APPEND FROM clientes DELIMITED
```

Tests

Status

Started

Es intención que **__dbDelim()** sea totalmente compatible con la función de CA-Clipper del mismo nombre y es la implementación subyacente de los comandos: APPEND FROM DELIMITED y COPY TO DELIMITED.

Platforms

Todas

Files

See Also:

[__dbSDF\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)

__dbSDF()

Copia el contenido de una base de datos a un archivo de texto SDF

Syntax

```
__dbSDF( <lExporta>, <xcArchivo>, [<aCampos>],  
[<bFor>], [<bWhile>], [<nNext>], [<nRegistro>], <lResto> ) --> NIL
```

Arguments

.F. agrega registros DESDE un archivo SDF.

<xcArchivo> es el nombre del archivo de texto del cual se va a copiar ó al que se va agregar datos. Si no es especificada una extensión. ".txt" es usada por defecto.

<aCampos> Un array con los nombres de los campos a los que se limita el proceso. Si no es especificado, ó si está vacío, entonces todos los campos son procesados.

<bFor> Un bloque de código opcional conteniendo una expresión FOR que reducirá el número de registros a ser procesados.

<bWhile> Un bloque de código opcional conteniendo una expresión WHILE que reducirá el número de registros a ser procesados.

<nNext> Si está presente, pero **<nRegistro>** no está presente, especifica procesar este número de registros, empezando con el registro actual. Un valor de cero significa no procesar registros.

<nRegistro> Si está presente, especifica el único registro a procesar un valor de cero significa no procesar ningún registro. Predomina sobre **<nNext>** y **<lResto>**.

<lResto> Si **<lExporta>** es .T., y es establecido en .T. y no hay argumentos **<nRegistro>**, **<nNext>** ó **<bWhile>**, entonces procesa todos los registros, desde el actual hasta el último.

Returns

Description

__dbSDF() copia todo ó el contenido seleccionado, desde una tabla de base de datos a un archivo de texto SDF ó, agrega todo ó el contenido seleccionado desde un archivo de texto SDF a una tabla de base de datos.

Examples

```
// Copia cuentas delinquentes dentro de un archivo de texto SDF  
USE cuentas NEW  
COPY TO deudas SDF FOR !EMPTY( cuentas->vencidas ) ;  
.AND. DATE() - cuentas->vencidas > 30  
  
// Importa nuevos registros de clientes  
USE clientes NEW  
APPEND FROM clientes SDF
```

Tests

Status

Started
Es intención que __dbSDF() sea totalmente compatible con la función de CA-Clipper del mismo nombre y es la implementación subyacente de los comandos: APPEND FROM SDF y COPY TO SDF.

Platforms

Todas

Files

See Also:

[__dbDelim\(\)](#)

[ARRAY\(\)](#)

[ARRAY\(\)](#)