Harbour Guide

ARRAY()

Create an uninitialized array of specified length

Syntax

ARRAY(<nElements> [, <nElements>...]) --> aArray

Arguments

<nElements> is the number of elements in the specified dimension.

Returns

<aArray> an array of specified dimensions.

Description

This function returns an uninitialized array with the length of <nElements>. Nested arrays are uninitialized within the same array pointer reference if additional parameters are specified. Establishing a memory variable with the same name as the array may destroy the original array and release the entire contents of the array. This depends, of course, on the data storage type of either the array or the variable with the same name as the array.

Examples

```
FUNCTION Main()
  LOCAL aArray:=Array(10)
  LOCAL x:=1
  FOR x:=1 to LEN(aArray)
    aArray[x]:=Array(x)
  NEXT
  Return Nil
```

Status

Ready

Compliance

This function is CA-CLIPPER Compliant in all Cases, except that arrays in Harbour can have an unlimited number of dimensions, while Clipper has a limit of 4096 array elements.

Files

Library is vm

See Also:

AADD()
ADEL()
AFILL()
AINS()

AADD()

Dynamically add an element to an array

Syntax

AADD(<aArray>[, <xValue>]) --> Value

Arguments

<aArray> The name of an array

<xValue> Element to add to array <aArray>

Returns

<Value> if specified <xValue>,<xValue> will return , otherwise this function returns a NIL value.

Description

This function dynamically increases the length of the array named <aArray> by one element and stores the value of <xValue> to that newly created element.

<xValue> may be an array reference pointer, which in turn may be stored to an array's subscript position.

Examples

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

Status

Ready

Files

Library is vm

See Also:

AINS()
ASIZE()

ASIZE()

Adjust the size of an array

Syntax

ASIZE(<aArray>, <nLen>) --> aTarget

Arguments

<aArray> Name of array to be dynamically altered

<nLen> Numeric value representing the new size of <aArray>

Returns

<aTarget> an array pointer reference to .

Description

This function will dynamically increase or decrease the size of <aArray> by adjusting the length of the array to <nLen> subscript positions.

If the length of the array <aArray> is shortened, those former subscript positions are lost. If the length of the array is lengthened a NIL value is assigned to the new subscript position.

Examples

```
aArray := { 1 }          // Result: aArray is { 1 }
ASIZE(aArray, 3)         // Result: aArray is { 1, NIL, NIL }
ASIZE(aArray, 1)         // Result: aArray is { 1 }
```

Status

Ready

Compliance

If HB_COMPAT_C53 is defined, the function generates an Error, else it will return the array itself.

Files

Library is vm

See Also:

AADD()
ADEL()
AFILL()
AINS()

ATAIL()

Returns the rightmost element of an array

Syntax

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

Arguments

<aArray> is the array.

Returns

<Element> the expression of the last element in the array.

Description

This function return the value of the last element in the array named <aArray>. This function does not alter the size of the array or any of the subscript values.

Examples

```
LOCAL array:= {"Harbour", "is", "Supreme", "Power"}
? ATAIL(aArray)
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

LEN()
ARRAY()
ASIZE()
AADD()

AINS()

Insert a NIL value at an array subscript position.

Syntax

AINS(<aArray>, <nPos>) --> aTarget

Arguments

<aArray> Array name.

<nPos> Subscript position in <aArray>

Returns

<aTarget> an array pointer reference.

Description

This function inserts a NIL value in the array named <aArray> at the <nPos>th position.

All array elements starting with the <nPos>th position will be shifted down one subscript position in the array list and the last item in the array will be removed completely. In other words, if an array element were to be inserted at the fifth subscript position, the element previously in the fifth position would now be located at the sixth position. The length of the array <aArray> will remain unchanged.

Examples

```
LOCAL aArray:={"Harbour","is","Power!","!!!"}
AINS(aArray,4)
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

AADD()
ACOPY()
ADEL()
AEVAL()
AFILL()
ASIZE()

ADEL()

Delete an element form an array.

Syntax

```
ADEL(<aArray>, <nPos>) --> aTarget
```

Arguments

<aArray>	Name of array from which an element is to be removed.
<nPos>	Subscript of the element to be removed.

Returns

<aTarget> an array pointer reference.

Description

This function deletes the element found at <nPos> subscript position in the array <aArray>. All elements in the array <aArray> below the given subscript position <nPos> will move up one position in the array. In other words, what was formerly the sixth subscript position will become the fifth subscript position. The length of the array <aArray> will remain unchanged,as the last element in the array will become a NIL data type.

Examples

```
LOCAL aArray
 aArray := { "Harbour","is","Power" }      // Result: aArray is

 ADEL(aArray, 2)            // Result: aArray is
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

ACOPY()
AINS()
AFILL()

AFILL()

Fill an array with a specified value

Syntax

AFILL(<aArray>, <xValue>, [<nStart>], [<nCount>]) --> aTarget

Arguments

<aArray>	Name of array to be filled.
<xValue>	Expression to be globally filled in <aArray>
<nStart>	Subscript starting position
<nCount>	Number of subscript to be filled

Returns

<aTarget> an array pointer.

Description

This function will fill each element of an array named <aArray> with the value <xValue>. If specified, <nStart> denotes the beginning element to be filled and the array elements will continue to be filled for <nCount> positions. If Not specified, the value of <nStart> will be 1, and the value of <nCount> will be the value of LEN(<aArray>); thus, all subscript positions in the array <aArray> will be filled with the value of <xValue>.

This function will work on only a single dimension of <aArray>. If there are array pointer references within a subscript <aArray>, those values will be lost, since this function will overwrite those values with new values.

Examples

```
LOCAL aTest:={Nil,0,1,2}
Afill(aTest,5)
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

AADD()
AEVAL()
DBSTRUCT()
ARRAY()

ASCAN()

Scan array elements for a specified condition

Syntax

ASCAN(<aTarget>, <xSearch>, [<nStart>], [<nCount>]) --> nStoppedAt

Arguments

<aTarget>	Name of array to be scanned.
<xSearch>	Expression to search for in <aTarget>
<nStart>	Beginning subscript position at which to start the search.
<nCount>	Number of elements to scan with <aTarget>.

Returns

<nStoppedAt> A numeric value of subscript position where <xSearch> was found.

Description

This function scan the content of array named <aTarget> for the value of <xSearch>. The return value is the position in the array <aTarget> in which <xSearch> was found. If it was not found, the return value will be 0.

If specified, the beginning subscript position at which to start scanning may be set with the value passed as <nStart>. The default is 1.

If specified, the number of array elements to scan may be set with the value passed as <nCount>. The default is the number of elements in the array <aTarget>.

If <xSearch> is a code block, the operation of the function is slightly different. Each array subscript pointer reference is passed to the code block to be evaluated. The scanning routine will continue until the value obtained from the code block is a logical true (.T.) or until the end of the array has been reached.

Examples

```
aDir:=Directory("*.prg")
AScan(aDir,,,{|x,y| x[1]="Test.prg"})
```

Status

Ready

Compliance

This function is not CA-Clipper compatible. Clipper ASCAN() is affected by the SET EXACT ON/OFF Condition

Files

Library is vm

See Also:

AEVAL()

AEVAL()

Evaluated the subscript element of an array

Syntax

AEVAL(<aArray>, <bBlock>, [<nStart>], [<nCount>]) --> aArray

Arguments

<aArray>	Is the array to be evaluated.
<bBlock>	Is a code block to evaluate for each element processed.
<nStart>	The beginning array element to evaluate.
<nCount>	The number of elements to process.

Returns

<aArray> an array pointer reference.

Description

This function will evaluate and process the subscript elements in <aArray>. A code block passed as <bBlock> defines the operation to be executed on each element of the array. All elements in <aArray> will be evaluated unless specified by a beginning subscript position in <nStart> for <nCount> elements.

Two parameters are passed to the code block <bBlock>. The individual elements in an array are the first parameter and the subscript position is the second.

AEVAL() does not replace a FOR...NEXT loop for processing arrays. If an array is an autonomous unit, AEVAL() is appropriate. If the array is to be altered or if elements are to be reevaluated, a FOR...NEXT loop is more appropriate.

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

EVAL()
DBEVAL()

ACOPY()

Copy elements from one array to another

Syntax

```
ACOPY( <aSource>, <aTarget>, [<nStart>], [<nCount>], [<nTargetPos>] )
--> aTarget
```

Arguments

<aSource> is the array to copy elements from.

<aTarget> is the array to copy elements to.

<nStart> is the beginning subscript position to copy from <aSource>

<nCount> the number of subscript elements to copy from <aSource>.

<nTargetPos> the starting subscript position in <aTarget> to copy elements to.

Returns

<aTarget> an array pointer reference

Description

This function copies array elements from <aSource> to <aTarget>. <nStart> is the beginning element to be copied from <aSource>; the default is 1.

<nCount> is the number of elements to be copied from <aSource>; the default is the entire array.

<nTargetPos> is the subscript number in the target array,<aTarget>, to which array elements are to be copied; the default is 1

This function will copy all data types in <aSource> to <aTarget>.

If an array element in <aSource> is a pointer reference to another array, that array pointer will be copied to <aTarget>; not all subdimensions will be copied from one array to the next. This must be accomplished via the ACLONE() function.

Note If array <aSource> is larger then <aTarget>, array elements will start copying at <nTargetPos> and continue copying until the end of array <aTarget> is reached. The ACOPY() function doesn't append subscript positions to the target array, the size of the target array <aTarget> remains constant.

Examples

```
LOCAL nCount := 2, nStart := 1, aOne, aTwo
aOne := {"HABOUR"," is ","POWER"}
aTwo := {"CLIPPER"," was ","POWER"}
ACOPY(aOne, aTwo, nStart, nCount)
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is vm

See Also:

ACLONE()
ADEL()
AEVAL()
AFILL()
AINS()
ASORT()

ACLONE()

Duplicate a multidimensional array

Syntax

ACLONE(<aSource>) --> aDuplicate

Arguments

<aSource> Name of the array to be cloned.

Returns

<aDuplicate> A new array pointer reference complete with nested array values.

Description

This function makes a complete copy of the array expressed as <aSource> and return a cloned set of array values.This provides a complete set of arrays values for all dimensions within the original array <aSource>

Examples

```
LOCAL aOne, aTwo
aOne := {"Harbour"," is ","POWER"}
aTwo := ACLONE(aOne)       // Result: aTwo is {1, 2, 3}
aOne[1] := "The Harbour Compiler"              // Result: aOne is {99, 2, 3}
                           // aTwo is still {1, 2, 3}
```

Status

Ready

Compliance

Clipper will return NIL if the parameter is not an array.

Files

Library is vm

See Also:

ACOPY()
ADEL()
AINS()
ASIZE()

ASORT()

Sort an array

Syntax

```
ASORT( <aArray>, [<nStart>], [<nCount>], [<bSort>] ) --> aArray
```

Arguments

<aArray> Array to be sorted.

<nStart> The first element to start the sort from, default is 1.

<nCount> Number of elements starting from <nStart> to sort, default is all elements.

<bSort> Code block for sorting order, default is ascending order {| x, y | x < y }. The code block should accept two parameters and must return .T. if the sort is in order, .F. if not.

Returns

<aArray> reference to the now sorted or NIL if the passed <aArray> is not an array.

Description

ASORT() sort all or part of a given array. If <bSort> is omitted, the function expect <aArray> to be one dimensional array containing single data type (one of: Character, Date, Logical, Numeric) and sort this array in ascending order: Character are sorted by their ASCII value, Dates are sorted chronologically, Logical put .F. values before .T., Numeric are sorted by their value.

If <bSort> is specified, it is used to handle the sorting order. With each time the block is evaluate, two array elements are passed to the code block, and <bSort> must return a logical value that state if those elements are in order (.T.) or not (.F.). Using this block you can sort multidimensional array, descending orders or even (but why would you want to do that) sort array that contain different data type.

Examples

```
// sort numeric values in ascending order
ASORT( { 3, 1, 4, 42, 5, 9 } )     // result: { 1, 3, 4, 5, 9, 42 }

// sort character strings in descending lexical order
aKeys := { "Ctrl", "Alt", "Delete" }
bSort := {| x, y | UPPER( x ) > UPPER( y ) }
ASORT( aKeys,,, bSort )      // result: { "Delete", "Ctrl", "Alt" }

// sort two-dimensional array according to 2nd element of each pair
aPair :=   { {"Sun",8}, {"Mon",1}, {"Tue",57}, {"Wed",-6} }
ASORT( aPair,,, {| x, y | x[2] < y[2] } )
// result: { {"Wed",-6}, {"Mon",1}, {"Sun",8}, {"Tue",57} }
```

Status

Ready

Compliance

Codeblock calling frequency and order differs from Clipper, since Harbour uses a different (faster) sorting algorithm (quicksort).

Files

Library is vm

See Also:

ASCAN()
EVAL()
ARRAY()

BIN2W()

Convert unsigned short encoded bytes into Harbour numeric

Syntax

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

Arguments

<cBuffer> is a character string that contain 16 bit encoded unsigned short integer (least significant byte first). The first two bytes are taken into account, the rest if any are ignored.

Returns

BIN2W() return numeric integer (or 0 if <cBuffer> is not a string).

Description

BIN2W() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. BIN2W() take two bytes of encoded 16 bit unsigned short integer and convert it into standard Harbour numeric value.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

BIN2W() is the opposite of W2BIN()

Examples

```
// Show header length of a DBF
FUNCTION main()
LOCAL nHandle, cBuffer := space( 2 )
nHandle := fopen( "test.dbf" )
IF nHandle > 0
   fseek( nHandle, 8 )
   fread( nHandle, @cBuffer, 2 )
   ? "Length of DBF header in bytes:", BIN2W( cBuffer )
   fclose( nHandle )
ELSE
   ? "Can not open file"
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2W() works exactly like CA-Clipper's BIN2W()

Files

Library is rtl

See Also:

- BIN2I()
- BIN2L()
- BIN2U()
- I2BIN()
- L2BIN()
- W2BIN()
- WORD()
- U2BIN()
- FREAD()

BIN2I()

Convert signed short encoded bytes into Harbour numeric

Syntax

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

Arguments

<cBuffer> is a character string that contain 16 bit encoded signed short integer (least significant byte first). The first two bytes are taken into account, the rest if any are ignored.

Returns

BIN2I() return numeric integer (or 0 if <cBuffer> is not a string).

Description

BIN2I() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. BIN2I() take two bytes of encoded 16 bit signed short integer and convert it into standard Harbour numeric value.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

BIN2I() is the opposite of I2BIN()

Examples

```
// Show DBF last update date
FUNCTION main()
LOCAL nHandle, cYear, cMonth, cDay
nHandle := fopen( "test.dbf" )
IF nHandle > 0
   fseek( nHandle, 1 )
   cYear := cMonth := cDay := " "
   fread( nHandle, @cYear , 1 )
   fread( nHandle, @cMonth, 1 )
   fread( nHandle, @cDay  , 1 )
   ? "Last update:", BIN2I( cYear ), BIN2I( cMonth ), BIN2I( cDay )
   fclose( nHandle )
ELSE
   ? "Can not open file"
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2I() works exactly like CA-Clipper's BIN2I()

Files

Library is rtl

See Also:

BIN2L()
BIN2U()
BIN2W()
I2BIN()
L2BIN()
W2BIN()
WORD()
U2BIN()

FREAD()

BIN2L()

Convert signed long encoded bytes into Harbour numeric

Syntax

BIN2L(<cBuffer>) --> nNumber

Arguments

<cBuffer> is a character string that contain 32 bit encoded signed long integer (least significant byte first). The first four bytes are taken into account, the rest if any are ignored.

Returns

BIN2L() return numeric integer (or 0 if <cBuffer> is not a string).

Description

BIN2L() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. BIN2L() take four bytes of encoded 32 bit signed long integer and convert it into standard Harbour numeric value.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

BIN2L() is the opposite of L2BIN()

Examples

```
// Show number of records in DBF
FUNCTION main()
LOCAL nHandle, cBuffer := space( 4 )
nHandle := fopen( "test.dbf" )
IF nHandle > 0
   fseek( nHandle, 4 )
   fread( nHandle, @cBuffer, 4 )
   ? "Number of records in file:", BIN2L( cBuffer )
   fclose( nHandle )
ELSE
   ? "Can not open file"
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2L() works exactly like CA-Clipper's BIN2L()

Files

Library is rtl

See Also:

- BIN2I()
- BIN2U()
- BIN2W()
- I2BIN()
- L2BIN()
- W2BIN()
- WORD()
- U2BIN()
- FREAD()

BIN2U()

Convert unsigned long encoded bytes into Harbour numeric

Syntax

BIN2U(<cBuffer>) --> nNumber

Arguments

<cBuffer> is a character string that contain 32 bit encoded unsigned long integer (least significant byte first). The first four bytes are taken into account, the rest if any are ignored.

Returns

BIN2U() return numeric integer (or 0 if <cBuffer> is not a string).

Description

BIN2U() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. BIN2U() take four bytes of encoded 32 bit unsigned long integer and convert it into standard Harbour numeric value.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

BIN2U() is the opposite of U2BIN()

Examples

```
// Show number of records in DBF
FUNCTION main()
LOCAL nHandle, cBuffer := space( 4 )
nHandle := fopen( "test.dbf" )
IF nHandle > 0
   fseek( nHandle, 4 )
   fread( nHandle, @cBuffer, 4 )
   ? "Number of records in file:", BIN2U( cBuffer )
   fclose( nHandle )
ELSE
   ? "Can not open file"
ENDIF
RETURN NIL
```

Status

Ready

Compliance

BIN2U() is an XBase++ compatibility function and does not exist as a standard CA-Clipper 5.x function. This function is only visible if source/rtl/binnum.c was compiled with the HB_COMPAT_XPP flag.

Files

Library is rtl

See Also:

BIN2I()
BIN2L()
BIN2W()
I2BIN()
L2BIN()
W2BIN()
WORD()
U2BIN()
FREAD()

I2BIN()

Convert Harbour numeric into signed short encoded bytes

Syntax

I2BIN(<nNumber>) --> cBuffer

Arguments

<nNumber> is a numeric value to convert (decimal digits are ignored).

Returns

I2BIN() return two bytes character string that contain 16 bit encoded signed short integer (least significant byte first).

Description

I2BIN() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. I2BIN() take a numeric integer value and convert it into two bytes of encoded 16 bit signed short integer.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

I2BIN() is the opposite of BIN2I()

Examples

```
// Update DBF "last update" date
#include "fileio.ch"
FUNCTION main()
LOCAL nHandle, cYear, cMonth, cDay
use test
? "Original update date is:", lupdate()
close
nHandle := fopen( "test.dbf", FO_READWRITE )
IF nHandle > 0
   fseek( nHandle, 1, )
   cYear  := I2BIN( 68 )
   cMonth := I2BIN(  8 )
   cDay   := I2BIN(  1 )
   fwrite( nHandle, cYear , 1 )   // write only the first byte
   fwrite( nHandle, cMonth, 1 )
   fwrite( nHandle, cDay  , 1 )
   fclose( nHandle )
   use test
   ? "New update date is:", lupdate()
   close
ELSE
   ? "Can not open file"
ENDIF
RETURN NIL
```

Status

Ready

Compliance

I2BIN() works exactly like CA-Clipper's I2BIN()

Files

Library is rtl

See Also:

BIN2I()
BIN2L()
BIN2U()

BIN2W()
L2BIN()
W2BIN()
WORD()
U2BIN()
FWRITE()

W2BIN()

Convert Harbour numeric into unsigned short encoded bytes

Syntax

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

Arguments

<nNumber> is a numeric value to convert (decimal digits are ignored).

Returns

W2BIN() return two bytes character string that contain 16 bit encoded unsigned short integer (least significant byte first).

Description

W2BIN() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. W2BIN() take a numeric integer value and convert it into two bytes of encoded 16 bit unsigned short integer.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

W2BIN() is the opposite of BIN2W()

Status

Ready

Compliance

W2BIN() is an XBase++ compatibility function and does not exist as a standard CA-Clipper 5.x function. This function is only visible if source/rtl/binnum.c was compiled with the HB_COMPAT_XPP flag.

Files

Library is rtl

See Also:

BIN2I()
BIN2L()
BIN2U()
BIN2W()
I2BIN()
L2BIN()
WORD()
U2BIN()
FWRITE()

L2BIN()

Convert Harbour numeric into signed long encoded bytes

Syntax

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

Arguments

<nNumber> is a numeric value to convert (decimal digits are ignored).

Returns

L2BIN() return four bytes character string that contain 32 bit encoded signed long integer (least significant byte first).

Description

L2BIN() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. L2BIN() take a numeric integer value and convert it into four bytes of encoded 32 bit signed long integer.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

L2BIN() is the opposite of BIN2L()

Status

Ready

Compliance

L2BIN() works exactly like CA-Clipper's L2BIN()

Files

Library is rtl

See Also:

- BIN2I()
- BIN2L()
- BIN2U()
- BIN2W()
- I2BIN()
- W2BIN()
- WORD()
- U2BIN()
- FWRITE()

U2BIN()

Convert Harbour numeric into unsigned long encoded bytes

Syntax

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

Arguments

<nNumber> is a numeric value to convert (decimal digits are ignored).

Returns

U2BIN() return four bytes character string that contain 32 bit encoded unsigned long integer (least significant byte first).

Description

U2BIN() is one of the low level binary conversion functions, those functions convert between Harbour numeric and a character representation of numeric value. U2BIN() take a numeric integer value and convert it into four bytes of encoded 32 bit unsigned long integer.

You might ask what is the need for such functions, well, first of all it allow you to read/write information from/to a binary file (like extracting information from DBF header), it is also a useful way to share information from source other than Harbour (C for instance).

U2BIN() is the opposite of BIN2U()

Status

Ready

Compliance

U2BIN() is an XBase++ compatibility function and does not exist as a standard CA-Clipper 5.x function. This function is only visible if source/rtl/binnum.c was compiled with the HB_COMPAT_XPP flag.

Files

Library is rtl

See Also:

BIN2I()
BIN2L()
BIN2U()
BIN2W()
I2BIN()
L2BIN()
W2BIN()
WORD()
FWRITE()

WORD()

Converts double to integer values.

Syntax

WORD(<nDouble>) --> <nInteger>

Arguments

<nDouble> is a numeric double value.

Returns

WORD() return an integer in the range +-32767

Description

This function converts double values to integers to use within the CALL command

Status

Ready

Compliance

The Clipper NG states that WORD() will only work when used in CALL commands parameter list, otherwise it will return NIL, in Harbour it will work anywhere.

Files

Library is rtl

See Also:

ARRAY()

DBEDIT()*

Browse records in a table

Syntax

DBEDIT([<nTop>], [<nLeft>], [<nBottom>], [<nRight>], [<acColumns>], [<xUserFunc>], [<xColumnSayPictures>], [<xColumnHeaders>], [<xHeadingSeparators>], [<xColumnSeparators>], [<xFootingSeparators>], [<xColumnFootings>]) --> lOk

Arguments

<nTop> coordinate for top row display. could range from 0 to MAXROW(), default is 0.

<nLeft> coordinate for left column display. could range from 0 to MAXCOL(), default is 0.

<nBottom> coordinate for bottom row display. could range from 0 to MAXROW(), default is MAXROW().

<nRight> coordinate for right column display. could range from 0 to MAXCOL(), default is MAXCOL().

<acColumns> is an array of character expressions that contain database fields names or expressions to display in each column. If not specified, the default is to display all fields from the database in the current work area.

<xUserFunc> is a name of a user defined function or a code block that would be called every time unrecognized key is been pressed or when there are no keys waiting to be processed and DBEDIT() goes into idle mode. If <xUserFunc> is a character string, it must contain root name of a valid user define function without parentheses. Both the user define function or the code block should accept two parameters: nMode, nCurrentColumn. Both should return a numeric value that correspond to one of the expected return codes (see table below for a list of nMode and return codes).

<xColumnSayPictures> is an optional picture. If is a character string, all columns would used this value as a picture string. If <xColumnSayPictures> is an array, each element should be a character string that correspond to a picture string for the column with the same index. Look at the help for @...SAY to get more information about picture values.

<xColumnHeaders> contain the header titles for each column, if this is a character string, all columns would have that same header, if this is an array, each element is a character string that contain the header title for one column. Header may be split to more than one line by placing semicolon (;) in places where you want to break line. If omitted, the default value for each column header is taken from <acColumns> or field name if <acColumns> was not specified.

<xHeadingSeparators> is an array that contain characters that draw the lines separating the headers and the fields data. Instead of an array you can use a character string that would be used to display the same line for all fields. Default value is a double line.

<xColumnSeparators> is an array that contain characters that draw the lines separating displayed columns. Instead of an array you can use a character string that would be used to display the same line for all fields. Default value is a single line.

<xFootingSeparators> is an array that contain characters that draw the lines separating the fields data area and the footing area. Instead of an array you can use a character string that would be used to display the same line for all footers. Default is to have to no footing separators.

<xColumnFootings> contain the footing to be displayed at the bottom of each column, if this is a character string, all columns would have that same footer, if this is an array, each element is a character string that contain the footer for one column. Footer may be split to more than one line by placing semicolon (;) in places where you want to break line. If omitted, no footer are displayed.

Returns

DBEDIT() return .F. if there is no database in use or if the number of columns to display is zero, else DBEDIT() return .T.

Description

DBEDIT() display and edit records from one or more work areas in a grid on screen. Each column is defined by element from <acColumns> and is the equivalent of one field. Each row is equivalent of one database record.

Following are active keys that handled by DBEDIT():

Key	Meaning
Left	Move one column to the left (previous field)
Right	Move one column to the right (next field)
Up	Move up one row (previous record)
Down	Move down one row (next record)
Page-Up	Move to the previous screen
Page-Down	Move to the next screen
Ctrl Page-Up	Move to the top of the file
Ctrl Page-Down	Move to the end of the file
Home	Move to the leftmost visible column
End	Move to the rightmost visible column
Ctrl Left	Pan one column to the left
Ctrl Right	Pan one column to the right
Ctrl Home	Move to the leftmost column
Ctrl End	Move to the rightmost column

When <xUserFunc> is omitted, two more keys are active:

Key	Meaning
Esc	Terminate BROWSE()
Enter	Terminate BROWSE()

When DBEDIT() execute <xUserFunc> it pass the following arguments: nMode and the index of current record in <acColumns>. If <acColumns> is omitted, the index number is the FIELD() number of the open database structure.

DBEDIT() nMode could be one of the following:

Dbedit.ch	Meaning
DE_IDLE	DBEDIT() is idle, all movement keys have been handled.
DE_HITTOP	Attempt to cursor past top of file.
DE_HITBOTTOM	Attempt to cursor past bottom of file.
DE_EMPTY	No records in work area, database is empty.
DE_EXCEPT	Key exception.

The user define function or code block must return a value that tell DBEDIT() what to do next.

User function return codes:

The user function is called once in each of the following cases: - The database is empty. - The user try to move past top of file or past bottom file. - Key exception, the uses had pressed a key that is not handled by DBEDIT(). - The keyboard buffer is empty or a screen refresh had just occurred DBEDIT() is a compatibility function, it is superseded by the TBrowse class and there for not recommended for new applications.

Examples

```
// Browse a file using default values
USE Test
DBEDIT()
```

Status

Started

Compliance

<xUserFunc> can take a code block value, this is a Harbour extension.

CA-Clipper will throw an error if there's no database open, Harbour would return .F.

CA-Clipper is buggy and will throw an error if the number of columns zero, Harbour would return .F.

The CA-Clipper 5.2 NG state that the return value is NIL, this is wrong and should be read logical.

There is an undocumented result code (3) from the user defined function in Clipper (both 87 and 5.x). This is an Append Mode which: "split the screen to allow data to be appended in windowed area". This mode is not supported by Harbour.

Files

Header files are dbedit.ch, inkey.ch Library is rtl

See Also:

@...SAY
BROWSE()
ARRAY()
TRANSFORM()

BROWSE()

Browse a database file

Syntax

BROWSE([<nTop>, <nLeft>, <nBottom>, <nRight>]) --> lOk

Arguments

<nTop> coordinate for top row display.

<nLeft> coordinate for left column display.

<nBottom> coordinate for bottom row display.

<nRight> coordinate for right column display.

Returns

BROWSE() return .F. if there is no database open in this work area, else it return .T.

Description

BROWSE() is a general purpose database browser, without any thinking you can browse a file using the following keys:

Key	Meaning
Left	Move one column to the left (previous field)
Right	Move one column to the right (next field)
Up	Move up one row (previous record)
Down	Move down one row (next record)
Page-Up	Move to the previous screen
Page-Down	Move to the next screen
Ctrl Page-Up	Move to the top of the file
Ctrl Page-Down	Move to the end of the file
Home	Move to the leftmost visible column
End	Move to the rightmost visible column
Ctrl Left	Pan one column to the left
Ctrl Right	Pan one column to the right
Ctrl Home	Move to the leftmost column
Ctrl End	Move to the rightmost column
Esc	Terminate BROWSE()

On top of the screen you see a status line with the following indication:

Record ###/###	Current record number / Total number of records.
<none>	There are no records, the file is empty.
<new>	You are in append mode at the bottom of file.
<Deleted>	Current record is deleted.
<bof>	You are at the top of file.

You should pass whole four valid coordinate, if less than four parameters are passed to BROWSE() the coordinate are default to: 1, 0, MAXROW(), MAXCOL().

Examples

```
// this one shows you how to browse around
USE Around
BROWSE()
```

Status

Started

Files

Library is rtl

See Also:

DBEDIT()*
ARRAY()

TBrowseDB()

Create a new TBrowse object to be used with database file

Syntax

TBrowseDB([<nTop>], [<nLeft>], [<nBottom>], [<nRight>]) --> oBrowse

Arguments

<nTop> coordinate for top row display.

<nLeft> coordinate for left column display.

<nBottom> coordinate for bottom row display.

<nRight> coordinate for right column display.

Returns

TBrowseDB() return new TBrowse object with the specified coordinate and a default :SkipBlock, :GoTopBlock and :GoBottomBlock to browse a database file.

Description

TBrowseDB() is a quick way to create a TBrowse object along with the minimal support needed to browse a database. Note that the returned TBrowse object contain no TBColumn objects and you need to add column for each field by your self.

Examples

```
for a good example, look at the source code for BROWSE() function
at source/rtl/browse.prg
```

Status

Started

Compliance

TBrowseDB() works exactly like CA-Clipper's TBrowseDB().

Files

Library is rtl

See Also:

BROWSE()
ARRAY()
ARRAY()
TBROWSENew()

dbSkipper()

Helper function to skip a database

Syntax

dbSkipper(<nRecs>) --> nSkipped

Arguments

<nRecs> is the number of records to skip relative to current record. Positive number would try to move the record pointer forward, while a negative number would try to move the record pointer back <nRecs> records.

Returns

dbSkipper() return the number of actual record skipped.

Description

dbSkipper() is a helper function used in browse mechanism to skip a number of records while giving the caller indication about the actual records skipped.

Examples

```
// open a file and find if we've got enough records in it
USE MonthSales
IF dbSkipper( 100 ) == 100
   ? "Good work! You can party now"
ELSE
   ? "Too bad, you should really work harder"
ENDIF
CLOSE
```

Status

Ready

Compliance

dbSkipper() is an XBase++ compatibility function and does not exist as a standard CA-Clipper 5.x function.

This function is only visible if source/rtl/browdb.prg was compiled with the HB_COMPAT_XPP flag.

Files

Library is rtl

See Also:

DBSKIP()
ARRAY()

Command line Utility

Compiler Options

Description

This spec goes for CLIPPERCMD, HARBOURCMD, Harbour compiler and #pragma directives in the source code.

The command line always overrides the envvar.

Note that some switches are not accepted in envvar,some others in #pragmas.

First the parser should start to step through all the tokens in the string separated by whitespace. (or just walk through all argv[])

1.) If the token begins with "-", it should be treated as a new style switch.

One or more switch characters can follow this. The "-" sign inside the token will turn off the switch.

If the switch has an argument all the following characters are treated as part of the argument.

The "/" sign has no special meaning here.

Switch	Result option
-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.) If the token begins with "/", it should be treated as a compatibility style switch.

The parser scans the token for the next "/" sign or EOS and treats the resulting string as one switch.

This means that a switch with an argument containing "/" sign has some limitations. This may be solved by allowing the usage of quote characters. This is mostly a problem on systems which use "/" as path separator.

The "-" sign has no special meaning here, it can't be used to disable a switch.

Switch	Result option
/w/n	(W N)
/wo/n	([invalid switch: wo] N)
/ihello/world/	(I=hello [invalid switch: world] [invalid switch: /])
/i"hello/world/"/w	(I=hello/world/ W)
/ihello\world\	(I=hello\world\ )

3.) If the token begins with anything else it should be skipped.

The Harbour switches are always case insensitive.

In the Harbour commandline the two style can be used together:
HARBOUR -wnes2 /gc0/q0 -ic:\hello

Exceptions:

- Handlig of the /CREDIT undocumented switch on Harbour command line is unusual, check the current code for this.

- The CLIPPER, HARBOUR and Harbour application command line parsing is a different beast, see CMDARG.C for a NOTE.

Notes:

- All occurences where a path is accepted, Harbour should handle the quote char to specify path containing space, negative sign, slash, or any other chars with special meaning.

```
/i"c:/hello/"
-i"c:/hello-n"
/i"c:/program files/"
-i"c:/program files/"
```

```
 Just some examples for the various accepted forms:
 //F20 == /F20 == F20 == F:20 == F20X
 //TMPPATH:C:\HELLO
 F20//TMPPATH:/TEMP///F:30000000 NOIDLE
 F0NOIDLEX10
 SQUAWKNOIDLE
```

"//" should always be used on the command line.

See Also:

Compiler Options

CLASS
Define a Class for Object Oriented Programming

Syntax

[CREATE] CLASS <ClassName> [<FROM, INHERIT> <SuperClass1> [,<SuperClassN>]] [STATIC]

Arguments

<ClassName> Name of the class to define. By tradition, Harbour classes start with "T" to avoid collisions with user- created classes.

<SuperClass1...n> The Parent class(es) to use for inheritance. Harbour supports Multiple Inheritance.

function. It will therefore not be available outside the current module.

Description

CLASS creates a class from which you can create objects. The CLASS command begins the class specification, in which the DATA elements (also known as instance variables) and METHODS of the class are named. The following scoping commands may also appear. They control the default scope of DATA and METHOD commands that follow them.

```
EXPORTED:
VISIBLE:
HIDDEN:
PROTECTED:
```

The class specification ends with the END CLASS command.

Classes can inherit from multiple <SuperClasses>, and the chain of inheritance can extend to many levels.

A program uses a Class by calling the Class Constructor, usually the New() method, to create an object. That object is usually assigned to a variable, which is used to access the DATA elements and methods.

Harbour's OOP syntax and implementation supports Scoping (Protect, Hidden and Readonly) and Delegating, and is largely compatible with Class(y)(tm), TopClass(tm) and Visual Objects(tm).

Examples

```
CLASS TBColumn

   DATA Block      // Code block to retrieve data for the column
   DATA Cargo      // User-definable variable
   DATA ColorBlock // Code block that determines color of data items
   DATA ColSep     // Column separator character
   DATA DefColor   // Array of numeric indexes into the color table
   DATA Footing    // Column footing
   DATA FootSep    // Footing separator character
   DATA Heading    // Column heading
   DATA HeadSep    // Heading separator character
   DATA Width      // Column display width
   DATA ColPos     // Temporary column position on screen

   METHOD New()    // Constructor

ENDCLASS
```

Status

Ready

Compliance

CLASS is a Harbour extension.

Platforms

All

See Also:

HBClass()
ARRAY()
DATA
METHOD

DATA

Alternate syntax for VAR: instance variable for the objects.

Syntax

```
DATA <DataName1> [,<DataNameN>] [ AS <type> ] [ INIT <uValue> ]
[[EXPORTED | VISIBLE] | [PROTECTED] | [HIDDEN]] [READONLY | RO]
```

Arguments

<DataName1> Name of the DATA

<type> Optional data type specification from the following: Character, Numeric, Date, Logical, Codeblock, Nil.

<uValue> Optional initial value when creating a new object.

outside of the class. VISIBLE is a synonym for EXPORTED.

within this class and its subclasses.

defined, and is not inherited by the subclasses.

clause, assignment is only permitted from the current class and its subclasses. If specified with the PROTECTED clause, assignment is only permitted from the current class. RO is a synonym for READONLY.

Description

DATA elements can also be thought of as the "properties" of an object. They can be of any data type, including codeblock. Once an object has been created, the DATA elements are referenced with the colon (:) as in MyObject:Heading := "Last name". Usually a class also defines methods to manipulate the DATA.

You can use the "AS <type>" clause to enforce that the DATA is maintained as a certain type. Otherwise it will take on the type of whatever value is first assigned to it.

Use the "INIT <uValue>" clause to initialize that DATA to <uValue> whenever a new object is created.

VAR can be a synonym for DATA, or it can use a slightly different syntax for compatibility with other dialects.

```
CLASS TBColumn

   DATA Block      // Code block to retrieve data for the column
   DATA Cargo      // User-definable variable
   DATA ColorBlock // Code block that determines color of data items
   DATA ColSep     // Column separator character
   DATA DefColor   // Array of numeric indexes into the color table
   DATA Footing    // Column footing
   DATA FootSep    // Footing separator character
   DATA Heading    // Column heading
   DATA HeadSep    // Heading separator character
   DATA Width      // Column display width
   DATA ColPos     // Temporary column position on screen

   METHOD New()    // Constructor

ENDCLASS
```

Status

Ready

Compliance

DATA is a Harbour extension.

Platforms

All

See Also:

ARRAY()
CLASS
METHOD
CLASSDATA
ARRAY()

CLASSDATA

Define a CLASSDATA variable for a class (NOT for an Object!)

Syntax

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

Arguments

<DataName1> Name of the DATA

<type> Optional data type specification from the following: Character, Numeric, Date, Logical, Codeblock, Nil

<uValue> Optional initial value at program startup

Description

CLASSDATA variables can also be thought of as the "properties" of an entire class. Each CLASSDATA exists only once, no matter how many objects are created. A common usage is for a counter that is incremented whenever an object is created and decremented when one is destroyed, thus monitoring the number of objects in existance for this class.

You can use the "AS <type>" clause to enforce that the CLASSDATA is maintained as a certain type. Otherwise it will take on the type of whatever value is first assigned to it. Use the "INIT <uValue>" clause to initialize that DATA to <uValue> whenever the class is first used.

Examples

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

Status

Ready

Compliance

CLASSDATA is a Harbour extension.

Platforms

All

See Also:

ARRAY()
CLASS
METHOD
DATA

METHOD

Declare a METHOD for a class in the class header

Syntax

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

Arguments

<MethodName>	Name of the method to define
<params,...>	Optional parameter list

Description

Methods are "class functions" which do the work of the class. All methods must be defined in the class header between the CLASS and ENDCLASS commands. If the body of a method is not fully defined here, the full body is written below the ENDCLASS command using this syntax:

```
METHOD <MethodName>( [<params,...>] ) CLASS <ClassName>
```

Methods can reference the current object with the keyword "Self:" or its shorthand version "::".

CLAUSES:

CONSTRUCTOR Defines a special method Class Constructor method, used to create objects. This is usually the New() method. Constructors always return the new object.

INLINE   Fast and easy to code, INLINE lets you define the code for the method immediately within the definition of the Class. Any methods not declared INLINE or BLOCK must be fully defined after the ENDCLASS command. The <Code,...> following INLINE receives a parameter of Self. If you need to receive more parameters, use the BLOCK clause instead.

BLOCK   Use this clause when you want to declare fast 'inline' methods that need parameters. The first parameter to <CodeBlock> must be Self, as in:

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

EXTERN   If an external function does what the method needs, use this clause to make an optimized call to that function directly.

SETGET   For calculated Data. The name of the method can be manipulated like a Data element to Set or Get a value.

VIRTUAL   Methods that do nothing. Useful for Base classes where the child class will define the method's behavior, or when you are first creating and testing a Class.

OPERATOR   Operator Overloading for classes. See example Tests/TestOp.prg for details.

CLASS <ClassName> Use this syntax only for defining a full method after the ENDCLASS command.

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
   ... <code> ...
   ... <code> ...
RETURN Self
```

Tests

```
TestOp.prg
```

Status

Ready

Compliance

METHOD is a Harbour extension.

Platforms

All

See Also:

- HBClass()
- ARRAY()
- DATA
- CLASS

MESSAGE

Route a method call to another Method

Syntax

```
MESSAGE <MessageName>   METHOD <MethodName>( [<params,...>] )
MESSAGE <MessageName>() METHOD <MethodName>( [<params,...>] )
```

Arguments

<MessageName>	The pseudo-method name to define
<MethodName>	The method to create and call when <MessageName> is invoked.
<params,...>	Optional parameter list for the method

Description

The MESSAGE command is a seldom-used feature that lets you re-route a call to a method with a different name. This can be necessary if a method name conflicts with a public function that needs to be called from within the class methods.

For example, your app may have a public function called BeginPaint() that is used in painting windows. It would also be natural to have a Window class method called :BeginPaint() that the application can call. But within the class method you would not be able to call the public function because internally methods are based on static functions (which hide public functions of the same name).

The MESSAGE command lets you create the true method with a different name (::xBeginPaint()), yet still allow the ::BeginPaint() syntax to call ::xBeginPaint(). This is then free to call the public function BeginPaint().

Examples

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

Status

Ready

Compliance

MESSAGE is a Harbour extension.

Platforms

All

See Also:

METHOD
DATA
CLASS
ARRAY()

ERROR HANDLER

Designate a method as an error handler for the class

Syntax

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

Arguments

<MethodName>	Name of the method to define
<params,...>	Optional parameter list

Description

ERROR HANDLER names the method that should handle errors for the class being defined.

Examples

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

Status

Ready

Compliance

ERROR HANDLER is a Harbour extension.

Platforms

All

See Also:

ARRAY()
ON ERROR
CLASS
METHOD
DATA

ON ERROR

Designate a method as an error handler for the class

Syntax

ON ERROR <MethodName>([<params,...>])

Arguments

<MethodName>	Name of the method to define
<params,...>	Optional parameter list

Description

ON ERROR is a synonym for ERROR HANDLER. It names the method that should handle errors for the class being defined.

Examples

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

Status

Ready

Compliance

ON ERROR is a Harbour extension.

Platforms

All

See Also:

ARRAY()
ERROR HANDLER
CLASS
METHOD
DATA

ENDCLASS

End the declaration of a class.

Syntax

```
ENDCLASS
```

Description

ENDCLASS marks the end of a class declaration. It is usually followed by the class methods that are not INLINE.

Examples

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

Status

Ready

Compliance

ON ERROR is a Harbour extension.

Platforms

All

See Also:

ARRAY()
CLASS
METHOD
DATA

Compiler Options

Compiler Options

Description

```
Invoking the Harbour compiler:
==============================

harbour <file[.prg]> [options]
or
harbour [options] <file[.prg]>
or
harbour [options] <file[.prg]> [options]

The command line options have to be separated by at least one space.  The
option can start with either '/' character or '-' character.

The Harbour command line options:
=================================

/a                automatic memvar declaration
=================

This causes all variables declared by PARAMETER, PRIVATE or PUBLIC
statements to be automatically declared as MEMVAR variables.

/b                debug info
=================

The compiler generates all information required for debugging

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

/es[<level>]      set exit severity
=================

/es or /es0 - all warnings are ignored and exit code returned by  the
compiler (accessed by DOS ERRORLEVEL command)  is equal to 0 if there are no errors
in compiled  source file.

/es1        - any warnings generate a non-zero exit code, but  output is
still created.

/es2        - all warnings are treated as errors and no output  file is
created. The exit code is set to a non-zero  value.

/g<type>          output type generated is <type>
=================

/gc[<type>]  output type: C source (.c) (default)  <type>: 0=compact
1=normal 2=verbose (default)

/go          output type: Platform dependant object module

/gh          output type: Harbour Portable Object (.hrb)


/i<path>          add #include file search path
=================

/k<mode>          enable <mode> compatibility mode
=================

/kc          clear all flags (strict Clipper compatibility)

/kh          Harbour extensions (default)

/ki          HB_INLINE support enabled (default)

/kr          use runtime settings for the macro compiler

/kx          other xbase dialects extensions (default)
```

```
/k?          invoke help information

/l                suppress line number information
=================

The compiler does not generate the source code line numbers in  the output
file. The PROCLINE() function will return 0 for  modules compiled using this
option.

/m                compile current module only
=================

/n                no implicit starting procedure
=================

The compiler does not create a procedure with the same name as  the
compiled file. This means that any declarations placed  before the first PROCEDURE
or FUNCTION statement have file-  wide scope and can be accessed/used in all
functions/procedures  defined in the compiled source file. All executable statements
placed at the beginning of the file and before the first  PROCEDURE/FUNCTION
statement are ignored.

/o<path>          output file drive and/or path
=================

/p                generate pre-processed output (.ppo) file
=================

The compiler only creates the file that contains the result of
pre-processing the source file.

/q                quiet
=================

The compiler does not print any messages during compiling  (except the
copyright info).

/q0     be really quiet and don't display even the copyright info

/r[<lib>]         request linker to search <lib> (or none)
=================

Currently not supported in Harbour.

/s                syntax check only
=================

The compiler checks the syntax only. No output file is generated.

/t<path>          path for temp file creation
=================

Currently not used in Harbour (the Harbour compiler does not  create any
temporary files).

/u[<file>]        use command definition set in <file> (or none)
=================

/v                variables are assumed M->
=================

All undeclared or unaliased variables are assumed MEMVAR  variables
(private or public variables). If this switch is not  used then the scope of such
variables is checked at runtime.

/w[<level>]       set warning level number (0..4, default 1)
=================

/w0         - no warnings

/w or /w1   - Clipper compatible warnings

/w2         - some useful warnings missed in Clipper

/w3         - warnings generated for Harbour language extensions  and also
enables strong type checking but only  warns against declared types, or types which
```

may be calculated at compile time

/w4   - Enables warning about suspicious operations, which means if you mix undeclared types, or types which can not be calculated at compile time,together with declared types, a warning will be generated.

/x[<prefix>]   set symbol init function name prefix (for .c only)
=================

Sets the prefix added to the generated symbol init function name (in C output currently). This function is generated automatically for every PRG module compiled. This additional prefix can be used to suppress problems with duplicated symbols during linking an application with some third party libraries.

/y   trace lex & yacc activity
=================

The Harbour compiler uses the FLEX and YACC utilities to parse the source code and to generate the required output file. This option traces the activity of these utilities.

/z   suppress logical shortcutting (.and. & .or.)
=================

Compilation in batch mode.
==========================

@<file>   compile list of modules in <file>
=================

Not supported yet.

Known incompatibilities between harbour and clipper compilers
===

NOTE:

If you want a 100% compatible runtime libraries then you have to define HARBOUR_STRICT_CLIPPER_COMPATIBILITY. This option should be defined in the file include/hbsetup.h (in fact this option is placed in a comment by default - you need to remove the /* */ characters only). This change has to be done before invoking the make utility.

Handling of undeclared variables
================================

When a value is assigned to an undeclared variable and the '-v' command line option is not used, then the Clipper compiler assumes that the variable is a PRIVATE or a PUBLIC variable and generates POPM (pop memvar) opcode.

When the value of an undeclared variable is accessed and the '-v' command line option is not used, the Clipper compiler generates PUSHV (push variable) opcode that determines the type of variable at runtime. If a field with the requested name exists in the current workarea then its value is used. If there is no field then a PRIVATE or a PUBLIC variable is used (if exists).

The Harbour compiler generates an opcode to determine the type of variable at runtime (POPVARIABLE or PUSHVARIABLE) in both cases (assignment and access).

The difference can be checked by the following code:

```
PROCEDURE MAIN()
PRIVATE myname

  DBCREATE( "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
```

Passing an undeclared variable by the reference
===

The Clipper compiler uses the special opcode PUSHP to pass a reference to an undeclared variable ('@' operator). The type of passed variable is checked at runtime (field or memvar). However, field variables cannot be passed by reference. This means that Clipper checks the memvar variable only and doesn't look for a field. This is the reason why the Harbour compiler uses the usual PUSHMEMVARREF opcode in such cases. Notice that the runtime behavior is the same in Clipper and in Harbour - only the generated opcodes are different.

Handling of object messages
===========================

The HARBOUR_STRICT_CLIPPER_COMPATIBILITY setting determines the way chained send messages are handled.

For example, the following code:

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

will be handled as:

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

in strict Clipper compatibility mode and

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

in non-strict mode.

In practice, Clipper will call the COUNT() function two times: the first time before addition and the second one after addition. In Harbour, COUNT() will be called only once, before addition.

The Harbour (non-strict) method is:
1) faster
2) it guarantees that the same instance variable of the same object will be changed

(See also: source/compiler/expropt.c)

Initialization of static variables
==================================

There is a difference in the initialization of static variables that are initialized with a codeblock that refers to a local variable. For example:

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

  MyLocalVar :=0
  ? EVAL( MyStaticVar )

RETURN
```

The above code compiles fine in Clipper, but it generates a runtime error
Error/BASE 1132 Bound error: array access
Called form (b)STATICS$(0)

In Harbour this code generates a compile time error: Error E0009 Illegal variable (b) initializer: 'MyLocalVar'

Both Clipper and Harbour are handling all local variables used in a codeblock
in a special way: they are detached from the local stack of function/procedure
where they are declared. This allows access to these variables after the exit from
a function/procedure. However, all static variables are initialized in a separate
procedure
('STATICS$' in Clipper and '(_INITSTATICS)' in Harbour) before the main
procedure and before all INIT procedures. The local variables don't exist on the
eval stack when static variables are initialized, so they cannot be detached.

CDOW()

Converts a date to the day of week

Syntax

CDOW(<dDate>) --> cDay

Arguments

<dDate>   Any date expression.

Returns

<cDay>   The current day of week.

Description

This function returns a character string of the day of the week, from a date expression <dDate> passed to it. If a NULL date is passed to the function, the value of the function will be a NULL byte.

Examples

```
? CDOW(DATE())
if CDOW(DATE()+10) =="SUNDAY"
   ? "This is a sunny day."
Endif
```

Status

Ready

Compliance

This function is Ca-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

DAY()
DOW()
DATE()
CMONTH()

CMONTH()

Return the name of the month.

Syntax

```
CMONTH(<dDate>)  --> cMonth
```

Arguments

<dDate> Any date expression.

Returns

<cMonth> The current month name

Description

This function returns the name of the month (January,February,etc.) from a date expression <dDate> passed to it. If a NULL date is passed to the function, the value of the function will be a NULL byte.

Examples

```
? CMONTH(DATE())
if CMONTH(DATE()+10) =="March"
   ? "Have you done your system BACKUP?"
Endif
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

- CDOW()
- DATE()
- MONTH()
- YEAR()
- DOW()
- DTOC()

DATE()

Return the Current OS Date

Syntax

```
DATE() --> dCurDate
```

Arguments

Returns

<dCurDate> Current system date.

Description

This function returns the current system date.

Examples

```
? Date()
```

Tests

```
? "Today is ",Day(date())," of ",cMonth(date())," of ",Year(date())
```

Status

Ready

Compliance

This function is Ca-Clipper Compliant

Platforms

All

Files

Library is rtl

See Also:

CTOD()
DTOS()
DTOC()
DAY()
MONTH()
CMONTH()

CTOD()

Converts a character string to a date expression

Syntax

```
CTOD(<cDateString>)  --> dDate
```

Arguments

<cDateString> A character date in format 'mm/dd/yy'

Returns

<dDate> A date expression

Description

This function converts a date that has been entered as a character expression to a date expression.The character expression will be in the form "MM/DD/YY" (based on the default value in SET DATE) or in the appropriate format specified by the SET DATE TO command. If an improper character string is passed to the function,an empty date value will be returned.

Examples

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

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

SET DATE
DATE()
DTOS()

DAY()

Return the numeric day of the month.

Syntax

```
DAY(<cDate>) --> nMonth
```

Arguments

<cDate> Any valid date expression.

Returns

<nMonth> Numeric value of the day of month.

Description

This function returns the numeric value of the day of month from a date.

Examples

```
? Day(DATE())
? Day(DATE()+6325)
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

- CTOD()
- DTOS()
- DTOC()
- DATE()
- MONTH()
- CMONTH()

DAYS()

Convert elapsed seconds into days

Syntax

DAYS(<nSecs>) --> nDay

Arguments

<nSecs> The number of seconds

Returns

<nDay> The number of days

Description

This function converts <nSecs> seconds to the equivalent number of days; 86399 seconds represents one day, 0 seconds being midnight.

Examples

```
? DAYS(2434234)
? "Has been passed ",DAYS(63251),' since midnight'
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

SECONDS()
SECS()
ELAPTIME()

DOW()

Value for the day of week.

Syntax

```
DOW(<dDate>) --> nDay
```

Arguments

<dDate> Any valid date expression

Returns

<nDay> The current day number

Description

This function returns the number representing the day of the week for the date expressed as <dDate>.

Examples

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

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

- DTOC()
- CDOW()
- DATE()
- DTOS()
- DAY()

DTOC()

Date to character conversion

Syntax

```
DTOC(<dDateString>)  --> cDate
```

Arguments

<dDateString> Any date

Returns

<dDate> Character represention of date

Description

This function converts any date expression (a field or variable) expressed as <dDateString> to a character expression in the default format "MM/DD/YY". The date format expressed by this function is controled in part by the date format specified in the SET DATE command

Examples

```
? DTOC(Date())
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

SET DATE
DATE()
DTOS()

DTOS()

Date to string conversion

Syntax

```
DTOS(<dDateString>)  --> cDate
```

Arguments

<dDateString> Any date

Returns

<dDate> String notation of the date

Description

This function returns the value of <dDateString> as a character string in the format of YYYYMMDD. If the value of <dDateString> is an empty date, this function will return eight blank spaces.

Examples

```
? DTOS(Date())
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

DTOC()
DATE()
DTOS()

ELAPTIME()

Calculates elapted time.

Syntax

ELAPTIME(<cStartTime>,<cEndTime>) --> cDiference

Arguments

<cStartTime> Start in time as a string format <cEndTime> End time as a string format

Returns

<cDiference> Difference between the times

Description

This function returns a string that shows the difference between the starting time represented as <cStartTime> and the ending time as <cEndTime>. If the stating time is greater then the ending time, the function will assume that the date changed once.

Examples

```
Static cStartTime
Init Proc Startup
cStartTime:=Time()

Exit Proc StartExit
? "You used this program by",ELAPTIME(cStartTime,Time())
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

SECS()
SECONDS()
TIME()
DAY()

MONTH()

Converts a date expression to a month value

Syntax

MONTH(<dDate>) --> nMonth

Arguments

<dDate> Any valid date expression

Returns

<nMonth> Corresponding number of the month in the year, ranging from 0 to 12

Description

This function returns a number that represents the month of a given date expression <dDate>. If a NULL date (CTOD('')) is passed to the function, the value of the function will be 0.

Examples

```
? Month(DATE())
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

CDOW()
DOW()
YEAR()
CMONTH()

SECONDS()

Returns the number of elapsed seconds past midnight.

Syntax

```
SECONDS() --> nSeconds
```

Arguments

Returns

<nSeconds> Number of seconds since midnight

Description

This function returns a numeric value representing the number of elapsed seconds based on the current system time. The system time is considered to start at 0 (midnight);it continues up to 86399 seconds.The value of the return expression is displayed in both seconds and hundredths of seconds.

Examples

```
? Seconds()
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

TIME()

SECS()

Return the number of seconds from the system date.

Syntax

```
SECS( <cTime> ) --> nSeconds
```

Arguments

<cTime> Character expression in a time string format

Returns

<nSeconds> Number of seconds

Description

This function returns a numeric value that is a number of elapsed seconds from midnight based on a time string given as <cTime>.

Examples

```
? Secs(Time())
? Secs(time()-10)
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

SECONDS()
ELAPTIME()
TIME()

TIME()

Returns the system time as a string

Syntax

```
TIME() --> cTime
```

Arguments

Returns

<cTime> Character string representing time

Description

This function returns the system time represented as a character expression in the format of HH:MM:SS

Examples

```
? Time()
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

DATE()
SECONDS()

YEAR()

Converts the year portion of a date into a numeric value

Syntax

YEAR(<cDate>) --> nYear

Arguments

<dDate> Any valid date expression

Returns

<nYear> The year portion of the date.

Description

This function returns the numeric value for the year in <dDate>. This value will always be a four-digit number and is not affected by the setting of the SET CENTURY and SET DATE commands. Addition ally, an empty date expression passed to this function will yield a zero value.

```
? Year(date())
? year(CTOD("01/25/3251"))
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

DAY()
MONTH()

__dbCopyStruct()

Create a new database based on current database structure

Syntax

__dbCopyStruct(<cFileName>, [<aFieldList>]) --> NIL

Arguments

<cFileName> is the name of the new database file to create. (.dbf) is the default extension if none is given.

<aFieldList> is an array where each element is a field name. Names could be specified as uppercase or lowercase.

Returns

__dbCopyStruct() always return NIL.

Description

__dbCopyStruct() create a new empty database file with a structure that is based on the currently open database in this work-area. If <aFieldList> is empty, the newly created file would have the same structure as the currently open database. Else, the new file would contain only fields that exactly match <aFieldList>.

__dbCopyStruct() can be use to create a sub-set of the currently open database, based on a given field list.

COPY STRUCTURE command is preprocessed into __dbCopyStruct() function during compile time.

Examples

```
// Create a new file that contain the same structure
 USE TEST
 __dbCopyStruct( "MyCopy.DBF" )

 // Create a new file that contain part of the original structure
 LOCAL aList
 USE TEST
 aList := { "NAME" }
 __dbCopyStruct( "OnlyName.DBF", aList )
```

Status

Ready

Compliance

__dbCopyStruct() works exactly like CA-Clipper's __dbCopyStruct()

Platforms

All

Files

Library is rdd

See Also:

- COPY STRUCTURE
- COPY STRUCTURE EXTENDED
- DBCREATE()
- DBSTRUCT()
- __dbCopyXStruct()
- __dbCreate()
- __dbStructFilter()

COPY STRUCTURE

Create a new database based on current database structure

Syntax

```
COPY STRUCTURE TO <xcFileName> [FIELDS <field,...>]
```

Arguments

TO <xcFileName></b> is the name of the new database file to create. (.dbf) is the default extension if none is given. It can be specified as a literal file name or as a character expression enclosed in parentheses.

FIELDS <field,...></b> is an optional list of field names to copy from the currently open database in the specified order, the default is all fields. Names could be specified as uppercase or lowercase.

Description

COPY STRUCTURE create a new empty database file with a structure that is based on the currently open database in this work-area.

COPY STRUCTURE can be use to create a sub-set of the currently open database, based on a given field list.

COPY STRUCTURE command is preprocessed into __dbCopyStruct() function during compile time.

Examples

```
// Create a new file that contains the same structure
 USE TEST
 COPY STRUCTURE TO MyCopy

 // Create a new file that contains part of the original structure
 USE TEST
 COPY STRUCTURE TO SomePart FIELDS name, address
```

Status

Ready

Compliance

COPY STRUCTURE works exactly as in CA-Clipper

Platforms

All

See Also:

COPY STRUCTURE EXTENDED
DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbCopyXStruct()
__dbCreate()
__dbStructFilter()

__dbCopyXStruct()

Copy current database structure into a definition file

Syntax

__dbCopyXStruct(<cFileName>) --> lSuccess

Arguments

<cFileName> is the name of target definition file to create. (.dbf) is the default extension if none is given.

Returns

__dbCopyXStruct() return (.F.) if no database is USED in the current work-area, (.T.) on success, or a run-time error if the file create operation had failed.

Description

__dbCopyXStruct() create a new database named <cFileName> with a pre-defined structure (also called "structure extended file"):

Field name	Type	Length	Decimals
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	3	0
FIELD_DEC	N	3	0

Each record in the new file contains information about one field in the original file. CREATE FROM could be used to create a database from the structure extended file.

For prehistoric compatibility reasons, Character fields which are longer than 255 characters are treated in a special way by writing part of the length in the FIELD_DEC according to the following formula (this is done internally):

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

Later if you want to calculate the length of a field you can use the following formula:

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

COPY STRUCTURE EXTENDED command is preprocessed into __dbCopyXStruct() function during compile time.

Examples

```
// Open a database, then copy its structure to a new file,
// Open the new file and list all its records
USE Test
__dbCopyXStruct( "TestStru" )
USE TestStru
LIST
```

Status

Ready

Compliance

__dbCopyXStruct() works exactly like CA-Clipper's __dbCopyXStruct()

Platforms

All

Files

Library is rdd

See Also:

COPY STRUCTURE
COPY STRUCTURE EXTENDED
CREATE
CREATE FROM
DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbCreate()

COPY STRUCTURE EXTENDED

Copy current database structure into a definition file

Syntax

```
COPY STRUCTURE EXTENDED TO <xcFileName>
```

Arguments

TO <xcFileName></b> The name of the target definition file to create. (.dbf) is the default extension if none is given. It can be specified as a literal file name or as a character expression enclosed in parentheses.

Description

COPY STRUCTURE EXTENDED create a new database named <cFileName> with a pre-defined structure (also called "structure extended file"):

Field name	Type	Length	Decimals
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	3	0
FIELD_DEC	N	3	0

Each record in the new file contains information about one field in the original file. CREATE FROM could be used to create a database from the structure extended file.

For prehistoric compatibility reasons, Character fields which are longer than 255 characters are treated in a special way by writing part of the length in the FIELD_DEC according to the following formula (this is done internally):

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

Later if you want to calculate the length of a field you can use the following formula:

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

COPY STRUCTURE EXTENDED command is preprocessed into __dbCopyXStruct() function during compile time.

Examples

```
// Open a database, then copy its structure to a new file,
// Open the new file and list all its records
USE Test
COPY STRUCTURE EXTENDED TO TestStru
USE TestStru
LIST
```

Status

Ready

Compliance

COPY STRUCTURE EXTENDED works exactly as in CA-Clipper

Platforms

All

See Also:

- COPY STRUCTURE
- CREATE
- CREATE FROM
- DBCREATE()
- DBSTRUCT()
- __dbCopyStruct()
- __dbCopyXStruct()
- __dbCreate()

__dbCreate()

Create structure extended file or use one to create new file

Syntax

```
__dbCreate( <cFileName>, [<cFileFrom>], [<cRDDName>], [<lNew>],
[<cAlias>] ) --> lUsed
```

Arguments

<cFileName> is the target file name to create and then open. (.dbf) is the default extension if none is given.

<cFileFrom> is an optional structure extended file name from which the target file <cFileName> is going to be built. If omitted, a new empty structure extended file with the name <cFileName> is created and opened in the current work-area.

<cRDDName> is RDD name to create target with. If omitted, the default RDD is used.

<lNew> is an optional logical expression, (.T.) opens the target file name <cFileName> in the next available unused work-area and makes it the current work-area. (.F.) opens the target file in the current work-area. Default value is (.F.). The value of <lNew> is ignored if <cFileFrom> is not specified.

<cAlias> is an optional alias to USE the target file with. If not specified, alias is based on the root name of <cFileName>.

Returns

__dbCreate() returns (.T.) if there is database USED in the current work-area (this might be the newly selected work-area), or (.F.) if there is no database USED. Note that on success a (.T.) would be returned, but on failure you probably end up with a run-time error and not a (.F.) value.

Description

__dbCreate() works in two modes depending on the value of <cFileFrom>:

1) If <cFileFrom> is empty or not specified a new empty structure extended file with the name <cFileName> is created and then opened in the current work-area (<lNew> is ignored). The new file has the following structure:

Field name	Type	Length	Decimals
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	3	0
FIELD_DEC	N	3	0

The CREATE command is preprocessed into the __dbCopyStruct() function during compile time and uses this mode.

2) If <cFileFrom> is specified, it is opened and assumed to be a structure extended file where each record contains at least the following fields (in no particular order): FIELD_NAME, FIELD_TYPE, FIELD_LEN and FIELD_DEC. Any other field is ignored. From this information the file <cFileName> is then created and opened in the current or new work-area (according to <lNew>), if this is a new work-area it becomes the current.

For prehistoric compatibility reasons, structure extended file Character fields which are longer than 255 characters should be treated in a special way by writing part of the length in the FIELD_DEC according to the following formula:

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

CREATE FROM command is preprocessed into __dbCopyStruct() function during compile time and use this mode.

Examples

```
// CREATE a new structure extended file, append some records and
// then CREATE FROM this file a new database file

__dbCreate( "template" )
DBAPPEND()
FIELD->FIELD_NAME := "CHANNEL"
FIELD->FIELD_TYPE := "N"
FIELD->FIELD_LEN  := 2
FIELD->FIELD_DEC  := 0
DBAPPEND()
FIELD->FIELD_NAME := "PROGRAM"
FIELD->FIELD_TYPE := "C"
FIELD->FIELD_LEN  := 20
FIELD->FIELD_DEC  := 0
DBAPPEND()
FIELD->FIELD_NAME := "REVIEW"
FIELD->FIELD_TYPE := "C"      // this field is 1000 char long
FIELD->FIELD_LEN  := 232      // 1000 % 256 = 232
FIELD->FIELD_DEC  := 3        // 1000 / 256 = 3
DBCLOSEAREA()
__dbCreate( "TV_Guide", "template" )
```

Status

Ready

Compliance

__dbCreate() works exactly as in CA-Clipper

Platforms

All

Files

Library is rdd

See Also:

COPY STRUCTURE
COPY STRUCTURE EXTENDED
CREATE
CREATE FROM
DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbCopyXStruct()

CREATE

Create empty structure extended file

Syntax

CREATE <xcFileName> [VIA <xcRDDName>] [ALIAS <xcAlias>]

Arguments

<xcFileName> is the target file name to create and then open. (.dbf) is the default extension if none is given. It can be specified as literal file name or as a character expression enclosed in parentheses.

VIA <xcRDDName></b> is RDD name to create target with. If omitted, the default RDD is used. It can be specified as literal name or as a character expression enclosed in parentheses.

ALIAS <xcAlias></b> is an optional alias to USE the target file with. If not specified, alias is based on the root name of <xcFileName>.

Description

CREATE a new empty structure extended file with the name <cFileName> and then open it in the current work-area. The new file has the following structure:

Field name	Type	Length	Decimals
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	3	0
FIELD_DEC	N	3	0

CREATE command is preprocessed into __dbCopyStruct() function during compile time and use this mode.

Examples

```
// CREATE a new structure extended file, append some records and
// then CREATE FROM this file a new database file

CREATE template
APPEND BLANK
FIELD->FIELD_NAME := "CHANNEL"
FIELD->FIELD_TYPE := "N"
FIELD->FIELD_LEN  := 2
FIELD->FIELD_DEC  := 0
APPEND BLANK
FIELD->FIELD_NAME := "PROGRAM"
FIELD->FIELD_TYPE := "C"
FIELD->FIELD_LEN  := 20
FIELD->FIELD_DEC  := 0
APPEND BLANK
FIELD->FIELD_NAME := "REVIEW"
FIELD->FIELD_TYPE := "C"      // this field is 1000 char long
FIELD->FIELD_LEN  := 232      // 1000 % 256 = 232
FIELD->FIELD_DEC  := 3        // 1000 / 256 = 3
CLOSE
CREATE TV_Guide FROM template
```

Status

Ready

Compliance

CREATE works exactly as in CA-Clipper

Platforms

All

See Also:

COPY STRUCTURE
COPY STRUCTURE EXTENDED
CREATE FROM
DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbCopyXStruct()
__dbCreate()

CREATE FROM

Create new database file from a structure extended file

Syntax

CREATE <xcFileName> FROM <xcFileFrom> [VIA <xcRDDName>] [NEW] [ALIAS <xcAlias>]

Arguments

<xcFileName> is the target file name to create and then open. (.dbf) is the default extension if none is given. It can be specified as literal file name or as a character expression enclosed in parentheses.

FROM <xcFileFrom></b> is a structure extended file name from which the target file <xcFileName> is going to be built. It can be specified as literal file name or as a character expression enclosed in parentheses.

VIA <xcRDDName></b> is RDD name to create target with. If omitted, the default RDD is used. It can be specified as literal name or as a character expression enclosed in parentheses.

NEW</b> open the target file name <xcFileName> in the next available unused work-area and making it the current work-area. If omitted open the target file in current work-area.

ALIAS <xcAlias></b> is an optional alias to USE the target file with. If not specified, alias is based on the root name of <xcFileName>.

Description

CREATE FROM open a structure extended file <xcFileFrom> where each record contain at least the following fields (in no particular order): FIELD_NAME, FIELD_TYPE, FIELD_LEN and FIELD_DEC. Any other field is ignored. From this information the file <xcFileName> is then created and opened in the current or new work-area (according to the NEW clause), if this is a new work-area it becomes the current.

For prehistoric compatibility reasons, structure extended file Character fields which are longer than 255 characters should be treated in a special way by writing part of the length in the FIELD_DEC according to the following formula:

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

CREATE FROM command is preprocessed into __dbCopyStruct() function during compile time and uses this mode.

Examples

See example in the CREATE command

Status

Ready

Compliance

CREATE FROM works exactly as in CA-Clipper

Platforms

All

See Also:

COPY STRUCTURE
COPY STRUCTURE EXTENDED
CREATE
DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbCopyXStruct()

__dbCreate()

__FLEDIT()*

Filter a database structure array

Syntax

```
__FLEDIT( <aStruct>, [<aFieldList>] ) --> aStructFiltered
```

Arguments

<aStruct> is a multidimensional array with database fields structure, which is usually the output from DBSTRUCT(), where each array element has the following structure:

<aFieldList> is an array where each element is a field name. Names could be specified as uppercase or lowercase.

Returns

__FLEDIT() return a new multidimensional array where each element is in the same structure as the original <aStruct>, but the array is built according to the list of fields in <aFieldList>. If <aFieldList> is empty, __FLEDIT() return reference to the original <aStruct> array.

Description

__FLEDIT() can be use to create a sub-set of a database structure, based on a given field list.

Note that field names in <aStruct> MUST be specified in uppercase or else no match would found.

SET EXACT has no effect on the return value.

__FLEDIT() is a compatibility function and it is synonym for __dbStructFilter() which does exactly the same.

Examples

```
LOCAL aStruct, aList, aRet
aStruct := { { "CODE",  "N",  4, 0 }, ;
             { "NAME",  "C", 10, 0 }, ;
             { "PHONE", "C", 13, 0 }, ;
             { "IQ",    "N",  3, 0 } }
aList := { "IQ", "NAME" }
aRet := __FLEDIT( aStruct, aList )
                  // { { "IQ", "N", 3, 0 }, { "NAME", "C", 10, 0 } }

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

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

aList := { "NOTEXIST" }
aRet := __FLEDIT( aStruct, aList )   // {}


// Create a new file that contain part of the original structure
LOCAL aStruct, aList, aRet
USE TEST
aStruct := DBSTRUCT()
aList := { "NAME" }
DBCREATE( "OnlyName.DBF", __FLEDIT( aStruct, aList ) )
```

Status

Ready

Compliance

CA-Clipper has internal undocumented function named __FLEDIT(), in Harbour we name it __dbStructFilter(). The new name gives a better description of what this function does. In Harbour __FLEDIT() simply calls __dbStructFilter() and therefor the later is the recommended function to use.

This function is only visible if source/rdd/dbstrux.prg was compiled with the HB_C52_UNDOC flag.

Platforms

All

Files

Header file is dbstruct.ch Library is rdd

See Also:

DBCREATE()
DBSTRUCT()
__dbCopyStruct()
__dbStructFilter()

__dbStructFilter()

Filter a database structure array

Syntax

__dbStructFilter(<aStruct>, [<aFieldList>]) --> aStructFiltered

Arguments

<aStruct> is a multidimensional array with database fields structure, which is usually the output from DBSTRUCT(), where each array element has the following structure:

<aFieldList> is an array where each element is a field name. Names could be specified as uppercase or lowercase.

Returns

__dbStructFilter() return a new multidimensional array where each element is in the same structure as the original <aStruct>, but the array is built according to the list of fields in <aFieldList>. If <aFieldList> is empty, __dbStructFilter() return reference to the original <aStruct> array.

Description

__dbStructFilter() can be use to create a sub-set of a database structure, based on a given field list.

Note that field names in <aStruct> MUST be specified in uppercase or else no match would be found.

SET EXACT has no effect on the return value.

Examples

```
LOCAL aStruct, aList, aRet
aStruct := { { "CODE",  "N",  4, 0 }, ;
             { "NAME",  "C", 10, 0 }, ;
             { "PHONE", "C", 13, 0 }, ;
             { "IQ",    "N",  3, 0 } }
aList := { "IQ", "NAME" }
aRet := __dbStructFilter( aStruct, aList )
                  // { { "IQ", "N", 3, 0 }, { "NAME", "C", 10, 0 } }

aRet := __dbStructFilter( aStruct, {} )
? aRet == aStruct // .T.

aList := { "iq", "NOTEXIST" }
aRet := __dbStructFilter( aStruct, aList )
                  // { { "IQ", "N", 3, 0 } }

aList := { "NOTEXIST" }
aRet := __dbStructFilter( aStruct, aList )   // {}


// Create a new file that contain part of the original structure
LOCAL aStruct, aList, aRet
USE TEST
aStruct := DBSTRUCT()
aList := { "NAME" }
DBCREATE( "OnlyName.DBF", __dbStructFilter( aStruct, aList ) )
```

Status

Ready

Compliance

__dbStructFilter() is a Harbour extension. CA-Clipper has an internal undocumented function named __FLEDIT() that does exactly the same thing. The new name gives a better description of what this function does.

Platforms

All

Files

Header file is dbstruct.ch Library is rdd

See Also:

- DBCREATE()
- DBSTRUCT()
- __dbCopyStruct()
- __FLEDIT()*

__Dir()*

Display listings of files

Syntax

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

Arguments

<cFileMask> File mask to include in the function return. It could contain path and standard wildcard characters as supported by your OS (like * and ?). If <cFileMask> contains no path, then SET DEFAULT path is used to display files in the mask.

Returns

__Dir() always returns NIL.

Description

If no <cFileMask> is given, __Dir() displays information about all *.dbf in the SET DEFAULT path. This information contains: file name, number of records, last update date and the size of each file.

If <cFileMask> is given, __Dir() list all files that match the mask with the following details: Name, Extension, Size, Date.

DIR command is preprocessed into __Dir() function during compile time.

__Dir() is a compatibility function, it is superseded by DIRECTORY() which return all the information in a multidimensional array.

Examples

```
__Dir()      // information for all DBF files in current directory

__Dir( "*.dbf" )         // list all DBF file in current directory

// list all PRG files in Harbour Run-Time library
// for DOS compatible operating systems
__Dir( "c:\harbour\source\rtl\*.prg" )

// list all files in the public section on a Unix like machine
__Dir( "/pub" )
```

Status

Ready

Compliance

DBF information: CA-Clipper displays 8.3 file names, Harbour displays the first 15 characters of a long file name if available.

File listing: To format file names displayed we use something like: PadR(Name, 8) + " " + PadR(Ext, 3) CA-Clipper use 8.3 file name, with Harbour it would probably cut long file names to feet this template.

Files

Library is rtl

See Also:

ADIR()
ARRAY()
SET DEFAULT
DIR

DIR
Display listings of files

Syntax

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

Arguments

<cFileMask> File mask to include in the function return. It could contain path and standard wildcard characters as supported by your OS (like * and ?). If <cFileMask> contains no path, then SET DEFAULT path is used to display files in the mask.

Description

If no <cFileMask> is given, __Dir() display information about all *.dbf in the SET DEFAULT path, this information contain: file name, number of records, last update date and the size of each file.

If <cFileMask> is given, __Dir() list all files that match the mask with the following details: Name, Extension, Size, Date.

DIR command is preprocessed into __Dir() function during compile time.

__Dir() is a compatibility function, it is superseded by DIRECTORY() which returns all the information in a multidimensional array.

Examples

```
DIR          // information for all DBF files in current directory

dir   "*.dbf"          // list all DBF file in current directory

// list all PRG files in Harbour Run-Time library
// for DOS compatible operating systems
  Dir  "c:\harbour\source\rtl\*.prg"

// list all files in the public section on a Unix like machine
  Dir  "/pub"
```

Status

Ready

Compliance

DBF information: CA-Clipper displays 8.3 file names, Harbour displays the first 15 characters of a long file name if available.

File listing: To format file names displayed we use something like: PadR(Name, 8) + " " + PadR(Ext, 3) CA-Clipper use 8.3 file name, with Harbour it would probably cut long file names to feet this template.

See Also:

ADIR()
ARRAY()
SET DEFAULT
__Dir()*

ADIR()

Fill pre-defined arrays with file/directory information

Syntax

ADIR([<cFileMask>], [<aName>], [<aSize>], [<aDate>], [<aTime>], [<aAttr>]) --> nDirEnries

Arguments

<cFileMask> File mask to include in the function return. It could contain path and standard wildcard characters as supported by your OS (like * and ?). If you omit <cFileMask> or if <cFileMask> contains no path, then the path from SET DEFAULT is used.

<aName> Array to fill with file name of files that meet <cFileMask>. Each element is a Character string and include the file name and extension without the path. The name is the long file name as reported by the OS and not necessarily the 8.3 uppercase name.

<aSize> Array to fill with file size of files that meet <cFileMask>. Each element is a Numeric integer for the file size in Bytes. Directories are always zero in size.

<aDate> Array to fill with file last modification date of files that meet <cFileMask>. Each element is of type Date.

<aTime> Array to fill with file last modification time of files that meet <cFileMask>. Each element is a Character string in the format HH:mm:ss.

<aAttr> Array to fill with attribute of files that meet <cFileMask>. Each element is a Character string, see DIRECTORY() for information about attribute values. If you pass array to <aAttr>, the function is going to return files with normal, hidden, system and directory attributes. If <aAttr> is not specified or with type other than Array, only files with normal attribute would return.

Returns

ADIR() return the number of file entries that meet <cFileMask>

Description

ADIR() return the number of files and/or directories that match a specified skeleton, it also fill a series of given arrays with the name, size, date, time and attribute of those files. The passed arrays should pre-initialized to the proper size, see example below. In order to include hidden, system or directories <aAttr> must be specified.

ADIR() is a compatibility function, it is superseded by DIRECTORY() which returns all the information in a multidimensional array.

Examples

```
LOCAL aName, aSize, aDate, aTime, aAttr, nLen, i
nLen := ADIR( "*.JPG" )     // Number of JPG files in this directory
IF nLen > 0
   aName := Array( nLen )   // make room to store the information
   aSize := Array( nLen )
   aDate := Array( nLen )
   aTime := Array( nLen )
   aAttr := Array( nLen )
   FOR i = 1 TO nLen
       ? aName[i], aSize[i], aDate[i], aTime[i], aAttr[i]
   NEXT
ELSE
   ? "This directory is clean from smut"
ENDIF
```

Status

Ready

Compliance

<aName> is going to be fill with long file name and not necessarily the 8.3 uppercase name.

Files

Library is rtl

See Also:

ARRAY()
ARRAY()
SET DEFAULT

DISKSPACE()

Get the amount of space available on a disk

Syntax

DISKSPACE([<nDrive>]) --> nDiskbytes

Arguments

<nDrive> The number of the drive you are requesting info on where 1 = A, 2 = B, etc. For 0 or no parameter, DiskSpace will operate on the current drive. The default is 0

Returns

<nDiskBytes> The number of bytes on the requested disk that match the requested type.

Description

By default, this function will return the number of bytes of free space on the current drive that is available to the user requesting the information.

If information is requested on a disk that is not available, a runtime error 2018 will be raised.

Examples

```
? "You can use : " +Str( DiskSpace() ) + " bytes " +;
Note: See tests\tstdspac.prg for another example
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

Dos,Win32,OS/2

Files

Library is rtl Header is fileio.ch

HB_DISKSPACE()

Get the amount of space available on a disk

Syntax

HB_DISKSPACE([<cDrive>] [, <nType>]) --> nDiskbytes

Arguments

<cDrive> The drive letter you are requesting info on. The default is A:

<nType> The type of space being requested.The default is HB_DISK_AVAIL.

Returns

<nDiskBytes> The number of bytes on the requested disk that match the requested type.

Description

By default, this function will return the number of bytes of free space on the current drive that is available to the user requesting the information.

There are 4 types of information available:

HB_FS_AVAIL   The amount of space available to the user making the request. This value could be less than HB_FS_FREE if disk quotas are supported by the O/S in use at runtime, and disk quotas are in effect. Otherwise, the value will be equal to that returned for HB_FS_FREE.

HB_FS_FREE   The actual amount of free diskspace on the drive.

HB_FS_USED   The number of bytes in use on the disk.

HB_FS_TOTAL   The total amount of space allocated for the user if disk quotas are in effect, otherwise, the actual size of the drive.

If information is requested on a disk that is not available, a runtime error 2018 will be raised.

Examples

```
? "You can use : " +Str( HB_DiskSpace() ) + " bytes " +;
  "Out of a total of " + Str( HB_DiskSpace('C:',HB_FS_TOTAL) )
```

Note: See tests\tstdspac.prg for another example

Status

Ready

Compliance

CA-Clipper will return an integer value which limits it's usefulness to drives less than 2 gigabytes. The Harbour version will return a floating point value with 0 decimals if the disk is > 2 gigabytes. <nType> is a Harbour extension.

Platforms

Dos,Win32,OS/2,Unix

Files

Library is rtl Header is fileio.ch

ERRORSYS()

Install default error handler

Syntax

ERRORSYS() --> NIL

Arguments

Returns

ERRORSYS() always return NIL.

Description

ERRORSYS() is called upon startup by Harbour and install the default error handler. Normally you should not call this function directly, instead use ERRORBLOCK() to install your own error handler.

Status

Ready

Compliance

ERRORSYS() works exactly like CA-Clipper's ERRORSYS().

Files

Library is rtl

See Also:

ARRAY()
ARRAY()

EVAL()

Evaluate a code block

Syntax

```
EVAL( <bBlock> [, <xVal> [,...]])   --> xExpression
```

Arguments

<bBlock>	Code block expression to be evaluated
<xVal>	Argument to be passed to the code block expression
<xVal...>	Argument list to be passed to the code block expression

Returns

<xExpression>	The result of the evaluated code block

Description

This function evaluates the code bloc expressed as <bBlock> and returns its evaluated value.If their are multiple expressions within the code block,the last expression will be value of this function.

If the code block requires parameters to be passed to it,they are specified in the parameter list <xVal> and following.Each parameter is separated by a comma within the expression list.

Examples

```
FUNC MAIN
LOCAL    sbBlock   := {|| NIL }
?  Eval( 1 )
?  Eval( @sbBlock )

? Eval( {|p1| p1 },"A","B")
? Eval( {|p1,p2| p1+p2 },"A","B")
? Eval( {|p1,p2,p3| p1 },"A","B")
Return Nil
```

Tests

See examples

Status

Ready

Compliance

This function is Ca Clipper compliant

Platforms

All

Files

Library is vm

See Also:

AEVAL()
DBEVAL()

FOPEN()

Open a file.

Syntax

```
FOPEN( <cFile>, [<nMode>] ) --> nHandle
```

Arguments

<cFile> Name of file to open.

<nMode> Dos file open mode.

Returns

<nHandle> A file handle.

Description

This function opens a file expressed as <cFile> and returns a file handle to be used with other low-level file functions. The value of <nMode> represents the status of the file to be opened; the default value is 0. The file open modes are as follows:

If there is an error in opening a file, a -1 will be returned by the function. Files handles may be in the range of 0 to 65535. The status of the SET DEFAULT TO and SET PATH TO commands has no effect on this function. Directory names and paths must be specified along with the file that is to be opened.

If an error has occured, see the returns values from FERROR() for possible reasons for the error.

Examples

```
IF (nH:=FOPEN('X.TXT',66) < 0
   ? 'File can't be opened'
ENDIF
```

Status

Ready
This function is CA-Clipper compliant

Files

Library is rtl Header is fileio.ch

See Also:

FCREATE()
FERROR()
FCLOSE()

FCREATE()

Creates a file.

Syntax

FCREATE(<cFile>, [<nAttribute>]) --> nHandle

Arguments

<cFile> is the name of the file to create.

<nAttribute> Numeric code for the file attributes.

Returns

<nHandle> Numeric file handle to be used in other operations.

Description

This function creates a new file with a filename of <cFile>. The default value of <nAttribute> is 0 and is used to set the attribute byte for the file being created by this function. The return value will be a file handle that is associated with the new file. This number will be between zero to 65,535, inclusive. If an error occurs, the return value of this function will be -1.

If the file <cFile> already exists, the existing file will be truncated to a file length of 0 bytes.

If specified, the following table shows the value for <nAttribute> and their related meaning to the file <cFile> being created by this function.

Examples

```
IF (nh:=FCREATE("TEST.TXT") <0
    ? "Cannot create file"
ENDIF
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rtl Header is fileio.ch

See Also:

FCLOSE()
FOPEN()
FWRITE()
FREAD()
FERROR()

FREAD()

Reads a specified number of bytes from a file.

Syntax

```
FREAD( <nHandle>, @<cBuffer>, <nBytes> ) --> nBytes
```

Arguments

<nHandle>	Dos file handle
<cBufferVar>	Character expression passed by reference.
<nBytes>	Number of bytes to read.

Returns

<nBytes> the number of bytes successfully read from the file. <nHandle>

Description

This function reads the characters from a file whose file handle is <nHandle> into a character memory variable expressed as <cBuffer>. The function returns the number of bytes successfully read into <cBuffer>.

The value of <nHandle> is obtained from either a call to the FOPEN() or the FCREATE() function.

The <cBuffer> expression is passed by reference and must be defined before this function is called. It also must be at least the same length as <nBytes>.

<nBytes> is the number of bytes to read, starting at the current file pointer position. If this function is successful in reading the characters from the file, the length of <cBuffer> or the number of bytes specified in <nBytes> will be the value returned. The current file pointer advances the number of bytes read with each successive read. The return value is the number of bytes successfully read from the file. If a 0 is returned, or if the number of bytes read matches neither the length of <cBuffer> nor the specified value in <nBytes> an end-of-file condition has been reached.

Examples

```
cBuffer:=SPACE(500)
IF (nH:=FOPEN('X.TXT))>0
   FREAD(Hh,@cBuffer,500)
   ? cbuffer
ENDIF
FCLOSE(nH)
```

Status

Ready

Compliance

This function is CA-Clipper compliant, but also extends the possible buffer size to strings greater than 65K (depending on platform).

Files

Library is rtl

See Also:

- BIN2I()
- BIN2L()
- BIN2W()
- FERROR()
- FWRITE()

FWRITE()

Writes characters to a file.

Syntax

FWRITE(<nHandle>, <cBuffer>, [<nBytes>]) --> nBytesWritten

Arguments

<nHandle>	DOS file handle number.
<cBuffer>	Character expression to be written.
<nBytes>	The number of bytes to write.

Returns

<nBytesWritten> the number of bytes successfully written.

Description

This function writes the contents of <cBuffer> to the file designated by its file handle <nHandle>. If used, <nBytes> is the number of bytes in <cBuffer> to write.

The returned value is the number of bytes successfully written to the DOS file. If the returned value is 0, an error has occurred (unless this is intended). A successful write occurs when the number returned by FWRITE() is equal to either LEN(<cBuffer>) or <nBytes>.

The value of <cBuffer> is the string or variable to be written to the open DOS file <nHandle>.

The value of <nBytes> is the number of bytes to write out to the file. The disk write begins with the current file position in <nHandle>. If this variable is not used, the entire contents of <cBuffer> is written to the file. To truncate a file. a call of FWRITE(nHandle, "", 0) is needed.

Examples

```
nHandle:=FCREATE('x.txt')
FOR X:=1 to 10
  FWRITE(nHandle,STR(x))
NEXT
FCLOSE(nHandle)
```

Status

Ready

Compliance

This function is not CA-Clipper compatile since it can writes strings greather the 64K

Files

Library is rtl

See Also:

FCLOSE()
FCREATE()
FERROR()
FOPEN()
I2BIN()
L2BIN()

FERROR()

Reports the error status of low-level file functions

Syntax

FERROR() --> <nErrorCode>

Returns

<nErrorCode> Value of the DOS error last encountered by a low-level file function.

FERROR() Return Values

Error	Meaning
0	Successful
2	File not found
3	Path not found
4	Too many files open
5	Access denied
6	Invalid handle
8	Insufficient memory
15	Invalid drive specified
19	Attempted to write to a write-protected disk
21	Drive not ready
23	Data CRC error
29	Write fault
30	Read fault
32	Sharing violation
33	Lock Violation

Description

After every low-level file function,this function will return a value that provides additional informationon the status of the last low-level file functions's performance.If the FERROR() function returns a 0, no error was detected.Below is a table of possibles values returned by the FERROR() function.

Examples

```
#include "Fileio.ch"
//
nHandle := FCREATE("Temp.txt", FC_NORMAL)
IF FERROR() != 0
   ? "Cannot create file, DOS error ", FERROR()
ENDIF
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Files

Library is rtl

See Also:

FCLOSE()
FERASE()
FOPEN()
FWRITE()

FCLOSE()

Closes an open file

Syntax

FCLOSE(<nHandle>) --> <lSuccess>

Arguments

<nHandle> DOS file handle

Returns

<lSuccess> Logical TRUE (.T.) or FALSE (.F.)

Description

This function closes an open file with a dos file handle of <nHandle> and writes the associated DOS buffer to the disk. The <nHandle> value is derived from the FCREATE() or FOPEN() function.

Examples

```
nHandle:=FOPEN('x.txt')
? FSEEK(nHandle0,2)
FCLOSE(nHandle)
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rtl

See Also:

FOPEN()
FCREATE()
FREAD()
FWRITE()
FERROR()

FERASE()

Erase a file from disk

Syntax

FERASE(<cFile>) --> nSuccess

Arguments

<cFile> Name of file to erase.

Returns

<nSuccess> 0 if successful, -1 if not

Description

This function deletes the file specified in <cFile> from the disk. No extensions are assumed. The drive and path my be included in <cFile>; neither the SET DEFAULT not the SET PATH command controls the performance of this function.If the drive or path is not used, the function will look for the file only on the currently selected direcytory on the logged drive.

If the function is able to successfully delete the file from the disk, the value of the function will be 0; otherwise a -1 will be returned.If not successfu, aditional information may be obtained by calling the FERROR() function.

Note: Any file to be removed by FERASE() must still be closed.

```
IF (FERASE("TEST.TXT")==0)
    ? "File successfully erased"
ELSE
    ? "File can not be deleted"
ENDIF
```

Status

Ready

Compliance

This function is CA-Clipper Compatible

Files

Library is rtl

See Also:

FERROR()
FRENAME()

FRENAME()

Renames a file

Syntax

FRENAME(<cOldFile>, <cNewFile>) --> nSuccess

Arguments

<cOldFile>	Old filenarne to he changed
<cNewFile>	New filename

Returns

<nSuccess> If sucessful, a 0 will he returned otherwise, a -1 will be returned.

Description

This function renames the specified file <cOldFile> to <cNewFile>. A filename and/or directory name may be specified for either para- meter. However, if a path is supplied as part of <cNewFile> and this path is different from either the path specified in <cOldFile> or (if none is used) the current drive and directory, the function will not execute successfully.

Neither parameter is subject to the control of the SET PATH TO or SET DEFAULT TO commands. In attempting to locate the file to be renamed, this function will search the default drive and directory or the drive and path specified in <cOldFile>. It will not search directories named by the SET PATH TO and SET DEFAULT TO commands or by the DOS PATH statement.

If the file specified in <cNewFile> exists or the file is open, the function will be unable to rename the file.If the function is unable to complete its operation,it will return a value of -1. If it is able to rename the file, the return value for the function will be 0.A call to FERROR() function will give additional infor- mation about any error found.

Examples

```
nResult:=FRENAME("x.txt","x1.txt")
IF nResult <0
   ? "File could not be renamed."
ENDIF
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rtl

See Also:

- ERASE
- FERASE()
- FERROR()
- FILE()
- RENAME

FSEEK()

Positions the file pointer in a file.

Syntax

FSEEK(<nHandle>, <nOffset>, [<nOrigin>]) --> nPosition

Arguments

<nHandle>	DOS file handle.
<nOffset>	The number of bytes to move.
<nOrigin>	The relative position in the file.

Returns

<nPosition> the current position relative to begin-of-file

Description

This function sets the file pointer in the file whose DOS file handle is <nHandle> and moves the file pointer by <expN2> bytes from the file position designated by <nOrigin>. The returned value is the relative position of the file pointer to the beginning-of-file marker once the operation has been completed.

<nHandle> is the file handle number. It is obtained from the FOPEN() or FCREATE() function.

The value of <nOffSet> is the number of bytes to move the file pointer from the position determined by <nOrigin>.The value of <nOffset> may be a negative number, suggesting backward movement.

The value of <nOrigin> designates the starting point from which the file pointer should he moved, as shown in the following table:

If a value is not provided for <nOrigin>, it defaults to 0 and moves the file pointer from the beginning of the file.

Examples

```
// here is a function that read one text line from an open file

// nH = file handle obtained from FOPEN()
// cB = a string buffer passed-by-reference to hold the result
// nMaxLine = maximum number of bytes to read

#define EOL HB_OSNEWLINE()
FUNCTION FREADln( nH, cB, nMaxLine )
LOCAL cLine, nSavePos, nEol, nNumRead
cLine := space( nMaxLine )
cB := ''
nSavePos := FSEEK( nH, 0, FS_RELATIVE )
nNumRead := FREAD( nH, @cLine, nMaxLine )
IF ( nEol := AT( EOL, substr( cLine, 1, nNumRead ) ) ) == 0
  cB := cLine
ELSE
  cB := SUBSTR( cLine, 1, nEol - 1 )
  FSEEK( nH, nSavePos + nEol + 1, FS_SET )
ENDIF
RETURN nNumRead != 0
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rtl Header is fileio.ch

See Also:

FCREATE()
FERROR()
FOPEN()
FREAD()
FREADSTR()
FWRITE()

FILE()

Tests for the existence of file(s)

Syntax

```
FILE( <cFileSpec> ) --> lExists
```

Arguments

<cFileSpec> Dos Skeleton or file name to find.

Returns

<lExists> a logical true (.T.) if the file exists or logical false (.F.).

Description

This function return a logical true (.T.) if the given filename <cFileSpec> exist.

Dos skeletons symbols may be used in the filename in <cFileSpec>, as may the drive and/or path name. If a path is not explicity specified, FILE() will look for the file in the SET DEFAULT path, then in each SET PATH path, until the file is found or there are no more paths to search. The DOS PATH is never searched and the current drive/directory is only searched if SET DEFAULT is blank.

Examples

```
? file('c:\harbour\doc\compiler.txt")
? file('c:/harbour/doc/subcodes.txt")
```

Status

S (wild card support is missing)

Compliance

This function is CA-Clipper compatible.

Files

Library is rtl

See Also:

SET DEFAULT
SET PATH
SET()

FREADSTR()

Reads a string from a file.

Syntax

FREADSTR(<nHandle>, <nBytes>) --> cString

Arguments

<nHandle>	DOS file handle number.
<nBytes>	Number of bytes to read.

Returns

<cString>	an characted expression

Description

This function returns a character string of <nBytes> bytes from a file whose DOS file handle is <nHandle>.

The value of the file handle <nHandle> is obtained from either the FOPEN() or FCREATE() functions.

The value of <nBytes> is the number of bytes to read from the file. The returned string will be the number of characters specified in <nBytes> or the number of bytes read before an end-of-file charac- ter (ASCII 26) is found.

NOTE This function is similar to the FREAD() function, except that it will not read binary characters that may he required as part of a header of a file construct. Characters Such as CHR(0) and CHR(26) may keep this function from performing its intended operation. In this event, the FREAD() function should he used in place of the FREADSTR() function.

Examples

```
IF ( nH := FOPEN("x.txt") ) > 0
   cStr := Freadstr(nH,100)
   ? cStr
ENDIF
FCLOSE(nH)
```

Status

Ready

Compliance

This function is not CA-Clipper compliant since may read strings greather the 65K depending of platform.

Files

Library is rtl

See Also:

BIN2I()
BIN2L()
BIN2W()
FERROR()
FREAD()
FSEEK()

RENAME

Changes the name of a specified file

Syntax

```
RENAME <cOldFile> TO <cNewFile>
```

Arguments

<cOldFile>	Old filename
<cNewFile>	New Filename

Description

This command changes the name of <cOldFile> to <cNewFile>.Both <cOldFile> and <cNewFile> must include a file extension.This command if not affected by the SET PATH TO or SET DEFAULT TO commands;drive and directoy designaters must be specified if either file is in a directory other then the default drive and directory.

If <cNewFile> id currently open or if it previously exists, this command will not perform the desired operation.

Examples

```
RENAME c:\autoexec.bat to c:\autoexec.old
```

Status

Ready

Compliance

This command is CA-Clipper compatible

Files

Library is rtl

See Also:

CURDIR()
ERASE
FILE()
FERASE()
FRENAME()

ERASE

Remove a file from disk

Syntax

```
ERASE <xcFile>
```

Arguments

<xcFile> Name of file to remove

Description

This command removes a file from the disk.The use of a drive,directo- ry,and wild-card skeleton operator is allowed for the root of the filename.The file extension is required.The SET DEFAULT and SET PATH commands do not affect this command.

The file must be considered closed by the operating system before it may be deleted.

Examples

```
Erase c:\autoexec.bat
Erase c:/temp/read.txt
```

Status

Ready

Compliance

This command is CA-Clipper compatible

See Also:

CURDIR()
FILE()
FERASE()
DELETE FILE

DELETE FILE

Remove a file from disk

Syntax

DELETE FILE <xcFile>

Arguments

<xcFile> Name of file to remove

Description

This command removes a file from the disk.The use of a drive,directo- ry,and wild-card skeleton operator is allowed for the root of the filename.The file extension is required.The SET DEFAULT and SET PATH commands do not affect this command.

The file must be considered closed by the operating system before it may be deleted.

Examples

```
Erase c:\autoexec.bat
Erase c:/temp/read.txt
```

Status

Ready

Compliance

This command is CA-Clipper compatible

See Also:

CURDIR()
FILE()
FERASE()
ERASE

__TYPEFILE()

Show the content of a file on the console and/or printer

Syntax

```
__TYPEFILE( <cFile>, [<lPrint>] ) --> NIL
```

Arguments

<cFile> is a name of the file to display. If the file have an extension, it must be specified (there is no default value).

<lPrint> is an optional logical value that specifies whether the output should go only to the screen (.F.) or to both the screen and printer (.T.), the default is (.F.).

Returns

__TYPEFILE() always return NIL.

Description

__TYPEFILE() function type the content of a text file on the screen with an option to send this information also to the printer. The file is displayed as is without any headings or formating.

If <cFile> contain no path, __TYPEFILE() try to find the file first in the SET DEFAULT directory and then in search all of the SET PATH directories. If <cFile> can not be found a run-time error occur.

Use SET CONSOLE OFF to suppress screen output. You can pause the output using Ctrl-S, press any key to resume.

__TYPEFILE() function is used in the preprocessing of the TYPE command.

Examples

```
The following examples assume a file name MyText.DAT exist in all
specified paths, a run-time error would displayed if it does not

// display MyText.DAT file on screen
__TYPEFILE( "MyText.DAT" )

// display MyText.DAT file on screen and printer
__TYPEFILE( "MyText.DAT", .T. )

// display MyText.DAT file on printer only
SET CONSOLE OFF
__TYPEFILE( "MyText.DAT", .T. )
SET CONSOLE ON
```

Status

Ready

Compliance

__TYPEFILE() works exactly like CA-Clipper's __TYPEFILE()

Files

Library is rtl

See Also:

COPY FILE
SET DEFAULT
SET PATH
SET PRINTER
TYPE

TYPE

Show the content of a file on the console, printer or file

Syntax

```
TYPE <xcFile> [TO PRINTER] [TO FILE <xcDestFile>]
```

Arguments

<xcFile> is a name of the file to display. If the file have an extension, it must be specified (there is no default value). It can be specified as literal file name or as a character expression enclosed in parentheses.

the screen and printer.

given (.txt) is added to the output file name. <xcDestFile> can be specified as literal file name or as a character expression enclosed in parentheses.

Description

TYPE command type the content of a text file on the screen with an option to send this information also to the printer or to an alternate file. The file is displayed as is without any headings or formating.

If <xcFile> contain no path, TYPE try to find the file first in the SET DEFAULT directory and then in search all of the SET PATH directories. If <xcFile> can not be found a run-time error occur.

If <xcDestFile> contain no path it is created in the SET DEFAULT directory.

Use SET CONSOLE OFF to suppress screen output. You can pause the output using Ctrl-S, press any key to resume.

Examples

```
The following examples assume a file name MyText.DAT exist in all
specified paths, a run-time error would displayed if it does not

// display MyText.DAT file on screen
TYPE MyText.DAT

// display MyText.DAT file on screen and printer
TYPE MyText.DAT TO PRINTER

// display MyText.DAT file on printer only
SET CONSOLE OFF
TYPE MyText.DAT TO PRINTER
SET CONSOLE ON

// display MyText.DAT file on screen and into a file MyReport.txt
TYPE MyText.DAT TO FILE MyReport
```

Status

Ready

Compliance

TYPE works exactly like CA-Clipper's TYPE

See Also:

- COPY FILE
- SET DEFAULT
- SET PATH
- SET PRINTER
- __TYPEFILE()

CURDIR()

Returns the current OS directory name.

Syntax

```
CURDIR( [<cDrive>] )  --> cPath
```

Arguments

<cDir> OS drive letter

Returns

<cPath> Name of directory

Description

This function yields the name of the current OS directory on a specified drive.If <cDrive> is not speficied,the currently logged drive will be used.

This function should not return the leading and trailing (back)slashes.

If an error has been detected by the function,or the current OS directory is the root,the value of the function will be a NULL byte.

Examples

```
? Curdir()
```

Status

Ready

Compliance

This function is Ca-Clipper Compatible

Platforms

ALL

Files

Library is rtl

See Also:

FILE()

COPY FILE

Copies a file.

Syntax

```
COPY FILE <cfile> TO <cfile1>
```

Arguments

<cFile> Filename of source file <cFile1> Filename of target file

Description

This command makes an exact copy of <cFile> and names it <cFile1>. Both files must have the file extension included; the drive and the directory names must also be specified if they are different from the default drive and/or director.<cFile1> also can refer to a OS device (e.g. LPT1).This command does not obsert the SET PATH TO or SET DEFAULT TO settings.

Examples

```
COPY FILE C:\HARBOUR\TESTS\ADIRTEST.PRG to C:\TEMP\ADIRTEST.PRG
COPY FILE c:\harbour\utils\hbdoc\gennf.prg to LPT1
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

ERASE
RENAME
FRENAME()
FERASE()

HB_FEOF()

Check for end-of-file.

Syntax

```
HB_FEOF( <nHandle> ) --> lIsEof
```

Arguments

<nHandle> The handle of an open file.

Returns

<lIsEof> .T. if the file handle is at end-of-file, otherwise .F.

Description

This function checks an open file handle to see if it is at E-O-F.

If the file handle is missing, not numeric, or not open, then this function returns .T. and sets the value returned by FERROR() to -1 (FS_ERROR) or a C-compiler dependent errno value (EBADF or EINVAL).

Examples

```
nH:=FOPEN('FILE.TXT')
? FREADSTR(nH,80)
IF HB_FEOF(nH)
   ? 'End-of-file reached.'
ELSE
   ? FREADSTR(nH,80)
ENDIF
```

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is rtl

See Also:

FERROR()

DIRREMOVE()

Attempt to remove an directory

Syntax

```
DIRCHANGE( <cDirectory> ) --> nError
```

Arguments

<cDirectory> The name of the directory you want to remove.

Returns

<nError> 0 if directory was successfully removed, otherwise the number of the last error.

Description

This function attempt to remove the specified directory in <cDirectory> If this function fail, the it will return the last OS error code number. See FERROR() function for the description of the error.

Examples

```
cDir:= ".\Backup"
if (DIRREMOVE(cDir)==0)
    ? "Remove of directory",cDir, "was successfull"
endif
```

Tests

See examples

Status

Ready

Compliance

This function is CA Clipper 5.3 compliant

Platforms

All

Files

Library is rtl

See Also:

MAKEDIR()
DIRCHANGE()
ISDISK()
ARRAY()
ARRAY()
FERROR()

DIRCHANGE()

Changes the directory

Syntax

DIRCHANGE(<cDirectory>) --> nError

Arguments

<cDirectory> The name of the directory you want do change into.

Returns

<nError> 0 if directory was successfully changed, otherwise the number of the last error.

Description

This function attempt to change the current directory to the one specidied in <cDirectory>.If this function fail, the it will return the last OS error code number.See FERROR() function for the description of the error.

Examples

```
if (DIRCHANGE("\temp")==0)
    ? "Change to diretory was successfull"
endif
```

Tests

See examples

Status

Ready

Compliance

This function is CA Clipper 5.3 compliant

Platforms

All

Files

Library is rtl

See Also:

MAKEDIR()
DIRREMOVE()
ISDISK()
ARRAY()
ARRAY()
FERROR()

MAKEDIR()

Create a new directory

Syntax

```
MAKEDIR( <cDirectory> ) --> nError
```

Arguments

<cDirectory> The name of the directory you want to create.

Returns

<nError> 0 if directory was successfully changed, otherwise the number of the last error.

Description

This function attempt to create a new directory with the name contained in <cDirectory>.If this function fail, the it will return the last OS error code number.See FERROR() function for the description of the error

Examples

```
cDir:= "Temp"
If (MAKEDIR( cDir)==0)
    ? "Directory ",cDir," successfully created
Endif
```

Tests

See examples

Status

Ready

Compliance

This function is CA Clipper 5.3 compliant

Platforms

All

Files

Library is rtl

See Also:

- DIRCHANGE()
- DIRREMOVE()
- ISDISK()
- ARRAY()
- ARRAY()
- FERROR()

ISDISK()

Verify if a drive is ready

Syntax

```
ISDISK( <cDrive> ) --> lSuccess
```

Arguments

<cDrive> An valid Drive letter

Returns

<lSuccess> .T. is the drive is ready, otherwise .F.

Description

This function attempts to access a drive. If the access to the drive was successfull, it will return true (.T.), otherwise false(.F.).This function is usefull for backup function, so you can determine if the drive that will recieve the backup data is ready or not.

Examples

```
IF ISDISK("A")
    ? "Drive is ready "
Endif
```

Tests

See Examples

Status

Ready

Compliance

This function is CA Clipper 5.3 compliant

Platforms

All

Files

Library is rtl

See Also:

DIRCHANGE()
MAKEDIR()
DIRREMOVE()
ARRAY()
ARRAY()

The Garbage Collector

Readme for Harbour Garbage Collect Feature

Description

The garbage collector uses the following logic: - first collect all memory allocations that can cause garbage; - next scan all variables if these memory blocks are still referenced.

Notice that only arrays, objects and codeblocks are collected because these are the only datatypes that can cause self-references (a[1]:=a) or circular references (a[1]:=b; b[1]:=c; c[1]:=a) that cannot be properly deallocated by simple reference counting.

Since all variables in harbour are stored inside some available tables (the eval stack, memvars table and array of static variables) then checking if the reference is still alive is quite easy and doesn't require any special treatment during memory allocation. Additionaly the garbage collector is scanning some internal data used by harbour objects implementation that also stores some values that can contain memory references. These data are used to initialize class instance variables and are stored in class shared variables.

In special cases when the value of a harbour variable is stored internally in some static area (at C or assembler level), the garbage collector will be not able to scan such values since it doesn't know their location. This could cause some memory blocks to be released prematurely. To prevent the premature deallocation of such memory blocks the static data have to store a pointer to the value created with hb_itemNew() function. Example: static HB_ITEM s_item; // this item can be released by the GC

static HB_ITEM_PTR pItem; // this item will be maintained correctly pItem = hb_itemNew(hb_param(1, IT_BLOCK));

However, scanning of all variables can be a time consuming operation. It requires that all allocated arrays have to be traversed through all their elements to find more arrays. Also all codeblocks are scanned for detached local variables they are referencing. For this reason, looking for unreferenced memory blocks is performed during the idle states.

The idle state is a state when there is no real application code executed. For example, the user code is stopped for 0.1 of a second during INKEY(0.1) - Harbour is checking the keyboard only during this time. It leaves however quite enough time for many other background tasks. One such background task can be looking for unreferenced memory blocks.

Allocating memory

The garbage collector collects memory blocks allocated with hb_gcAlloc() function calls. Memory allocated by hb_gcAlloc() should be released with hb_gcFree() function.

The garbage collecting

During scanning of unreferenced memory the GC is using a mark & sweep algorithm. This is done in three steps:

1) mark all memory blocks allocated by the GC with unused flag;

2) sweep (scan) all known places and clear unused flag for memory blocks that are referenced there;

3) finalize collecting by deallocation of all memory blocks that are still marked as unused and that are not locked.

To speed things up, the mark step is simplified by swapping the meaning of the unused flag. After deallocation of unused blocks all still alive memory blocks are marked with the same 'used' flag so we can reverse the meaning of this flag to 'unused' state in the next collecting. All new or unlocked memory blocks are automatically marked as 'unused' using the current flag, which assures that all memory blocks are marked with the same flag before the sweep step will start. See hb_gcCollectAll() and hb_gcItemRef()

Calling the garbage collector from harbour code

```
      ---------------------------------------------

      The garbage collector can be called directly from the harbour code.  This is
      usefull in situations where there is no idle states available  or the application
      is working in the loop with no user interaction  and there is many memory
      allocations.  See HB_GCALL() for explanation of how to call this function from your
      harbour code.
```

See Also:

hb_gcAlloc()
hb_gcFree()
hb_gcCollectAll()
hb_gcItemRef()
HB_GCALL()
hb_idleState()

hb_gcAlloc()

Allocates memory that will be collected by the garbage collector.

Syntax

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

Arguments

<ulSize> Requested size of memory block

<pCleanupFunc> Pointer to HB_GARBAGE_FUNC function that will be called directly before releasing the garbage memory block or NULL. This function should release all other memory allocated and stored inside the memory block. For example, it releases all items stored inside the array. The functions receives a single parameter: the pointer to memory allocated by hb_gcAlloc().

Returns

Description

hb_gcAlloc() is used to allocate the memory that will be tracked by the garbage collector. It allows to properly release memory in case of self-referencing or cross-referencing harbour level variables. Memory allocated with this function should be released with hb_gcFree() function or it will be automatically deallocated by the GC if it is not locked or if it is not referenced by some harbour level variable.

Examples

See source/vm/arrays.c

Status

Clipper

Compliance

This function is a Harbour extension

Platforms

All

Files

source/vm/garbage.c

See Also:

hb_gcFree()

hb_gcFree()

Releases the memory that was allocated with hb_gcAlloc().

Syntax

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

Arguments

<pMemoryPtr> The pointer to memory for release. This memory pointer have to be allocated with hb_gcAlloc() function.

Returns

Description

hb_gcFree() is used to deallocate the memory that was allocated with the hb_gcAlloc() function.

Examples

```
See source/vm/arrays.c
```

Status

Clipper

Compliance

This function is a Harbour extension

Platforms

All

Files

source/vm/garbage.c

See Also:

hb_gcAlloc()

hb_gcCollectAll()

Scans all memory blocks and releases the garbage memory.

Syntax

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

Arguments

Returns

Description

This function scans the eval stack, the memvars table, the array of static variables and table of created classes for referenced memory blocks. After scanning all unused memory blocks and blocks that are not locked are released.

Status

Clipper

Compliance

This function is a Harbour extension

Platforms

All

Files

source/vm/garbage.c

See Also:

hb_gcAlloc()
hb_gcFree()

hb_gcItemRef()

Marks the memory to prevent deallocation by the garbage collector.

Syntax

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

Arguments

<pItem> The pointer to item structure that will be scanned. The passed item can be of any datatype although arrays, objects and codeblocks are scanned only. Other datatypes don't require locking so they are simply ignored.

Returns

Description

The garbage collector uses hb_gcItemRef() function during scanning of referenced memory pointers. This function checks the type of passed item and scans recursively all other memory blocks referenced by this item if it is an array, an object or a codeblock.

NOTE: This function is reserved for the garbage collector only. It cannot be called from the user code - calling it can cause unpredicted results (memory blocks referenced by the passed item can be released prematurely during the closest garbage collection).

Status

Clipper

Compliance

This function is a Harbour extension

Platforms

All

Files

source/vm/garbage.c

See Also:

hb_gcAlloc()
hb_gcFree()

HB_GCALL()

Scans the memory and releases all garbage memory blocks.

Syntax

HB_GCALL()

Arguments

Returns

Description

This function releases all memory blocks that are considered as the garbage.

Status

Harbour

Compliance

This function is a Harbour extension

Platforms

All

Files

source/vm/garbage.c

See Also:

hb_gcCollectAll()

GNU License

Gnu License File Part 1

Description

See Also:

GNU License Part 2

GNU License Part 2

Gnu License File Part 2

Description

See Also:

Harbour Extensions

Harbour Extensions

Description

Language extensions:
----

* Class generation and management.

Clipper only allowed creation of objects from a few standard classes.

In Harbour, you can create your own classes--complete with Methods, Instance Variables, Class Variables and Inheritance. Entire applications can be designed and coded in Object Oriented style.

* @<FunctionName>()

Returns the pointer (address) to a function.

The returned value is not useful to application-level programming, but is used at a low level to implement object oriented coding. (Internally, a class method is a static function and there is no symbol for it, so it is accessed via its address).

* Class HBGetList

Object oriented support for GetLists management.

* ProcName() support for class Method names.

Class Methods can be retrieved from the call stack.

* Memory() has new return values.

See hbmemory.ch

* Transform() --> new function in format string

@0   Make a zero padded string out of the number.

* SToD() --> dDate

New function that converts a yyyymmdd string to a Date value.

* Optional Compile Time STRONG TYPE declaration (and compile time TYPE MISMATCH warnings)

Example: LOCAL/STATIC Var AS ...

* The Harbour debugger provides new interesting classes:

- Class TDbWindow could be the foundation for a generic multiplatform

- Class TForm

- Class TDbMenu implement both pulldown and popup menus.

RTL enhanced functionality:

- Directory(<cMask>, <cFlags>, <lEightDotThree>)

The 3rd parameter is a Harbour (optional) parameter and indicates that on those platforms that support long filenames, that you wish to receive what would be considered the dos equivalant 8.3 name. Could affect Adir() and Dir if they were modified to take advantage of it - currently, they will return long names if the os supports it.

- HB_DiskSpace(<nDrive>, <nType>)

The second parameter is a Harbour (optional) parameter and indicates the type of diskinfo being requested. See en/diskspac.txt for info.

hb_parc()

Retrieve a string parameter

Syntax

C Prototype

```
#include <hbapi.h>
hb_parc( int iParam, ... ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parclen()

Retrieve a string parameter length

Syntax

C Prototype

```
#include <hbapi.h>
hb_parclen( int iParam, ... ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parcsiz()

Retrieve a by-reference string parameter length, including terminator

Syntax

C Prototype

```
#include <hbapi.h>
hb_parcsiz( int iParam, ... ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_pards()

Retrieve a date as a string yyyymmdd

Syntax

C Prototype

```
#include <hbapi.h>
hb_pards( int iParam, ... ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_pardsbuff()

Retrieve a date as a string yyyymmdd

Syntax

C Prototype

```
#include <hbapi.h>
hb_pardsbuff( char * szDate, int iParam, ... ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parinfa()

Retrieve length or element type of an array parameter

Syntax

C Prototype

```
#include <hbapi.h>
hb_parinfa( int iParamNum, ULONG uiArrayIndex ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parinfo()

Determine the param count or data type

Syntax

C Prototype

```
#include <hbapi.h>
hb_parinfo( int iParam ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parl()

Retrieve a logical parameter as an int

Syntax

C Prototype

```
#include <hbapi.h>
hb_parl( int iParam, ... ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parnd()

Retrieve a numeric parameter as a double

Syntax

C Prototype

```
#include <hbapi.h>
hb_parnd( int iParam, ... ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parni()

Retrieve a numeric parameter as a integer

Syntax

C Prototype

```
#include <hbapi.h>
hb_parni( int iParam, ... ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_parnl()

Retrieve a numeric parameter as a long

Syntax

C Prototype

```
#include <hbapi.h>
hb_parnl( int iParam, ... ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_param()

Retrieve a direct pointer to an item parameter

Syntax

C Prototype

```
#include <hbapi.h>
hb_param( int iParam, int iMask ) --> ( PHB_ITEM ) pResult
```

Arguments

<iParam> The 1-based parameter to retrieve.

Returns

hb_param() returns a direct pointer to an item on the eval stack.

Description

This item will be removed (set to NIL) after a function cleanup, so if the item needs to survive the current function (e.g. copied to a static) you should use hb_itemParam instead.

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

See Also:

hb_itemParam()

hb_pcount()

Returns the number of supplied parameters

Syntax

C Prototype

```
#include <hbapi.h>
hb_pcount( void ) --> ( int )iResult
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_pcount() --> ((int) hb_stack.pBase->item.asSymbol.paramcnt)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_ret()

Post a NIL return value

Syntax

C Prototype

```
#include <hbapi.h>
hb_ret( void ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_ret() --> hb_itemClear(&hb_stack.Return)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retc()
Returns a string

Syntax

C Prototype

```
#include <hbapi.h>
hb_retc( char * szText ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retc(szText) --> hb_itemPutC(&hb_stack.Return, szText)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retclen()

Returns a string with a specific length

Syntax

C Prototype

```
#include <hbapi.h>
hb_retclen( char * szText, ULONG ulLen ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retclen(szText, ulLen) --> hb_itemPutCL(&hb_stack.Return, szText, ulLen)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retds()

Returns a date, must use yyyymmdd format

Syntax

C Prototype

```
#include <hbapi.h>
hb_retds( char * szDate ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retds(szDate) --> hb_itemPutDS(&hb_stack.Return, szDate)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retd()

Returns a date

Syntax

```
C Prototype

#include <hbapi.h>
hb_retd( long lYear, long lMonth, long lDay ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retd(lYear, lMonth, lDay) --> hb_itemPutD(&hb_stack.Return, lYear, lMonth, lDay)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retdl()

Returns a long value as a julian date

Syntax

C Prototype

```
#include <hbapi.h>
hb_retdl( long lJulian ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retdl(lJulian) --> hb_itemPutDL(&hb_stack.Return, lJulian)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retl()

Returns a logical integer

Syntax

C Prototype

```
#include <hbapi.h>
hb_retl( int iTrueFalse ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retl(iLogical) --> hb_itemPutL(&hb_stack.Return, iLogical ? TRUE : FALSE)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retnd()

Returns a double

Syntax

C Prototype

```
#include <hbapi.h>
hb_retnd( double dNumber ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retnd(dNumber) --> hb_itemPutND(&hb_stack.Return, dNumber)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retni()

Returns a integer number

Syntax

C Prototype

```
#include <hbapi.h>
hb_retni( int iNumber ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retni(iNumber) --> hb_itemPutNI(&hb_stack.Return, iNumber)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retnl()

Returns a long number

Syntax

```
C Prototype

#include <hbapi.h>
hb_retnl( long lNumber ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retnl(lNumber) --> hb_itemPutNL(&hb_stack.Return, lNumber)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retnlen()

Returns a double, with specific width and decimals

Syntax

C Prototype

```
#include <hbapi.h>
hb_retnlen( double dNumber, int iWidth, int iDec ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retnlen(dNumber, iWidth, iDec) --> hb_itemPutNLen(&hb_stack.Return, dNumber, iWidth, iDec)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retndlen()

Returns a double, with specific width and decimals

Syntax

C Prototype

```
#include <hbapi.h>
hb_retndlen( double dNumber, int iWidth, int iDec ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retndlen(dNumber, iWidth, iDec) --> hb_itemPutNDLen(&hb_stack.Return, dNumber, iWidth, iDec)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retnilen()

Returns a integer number, with specific width

Syntax

C Prototype

```
#include <hbapi.h>
hb_retnilen( int iNumber, int iWidth ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retnilen(iNumber, iWidth) --> hb_itemPutNILen(&hb_stack.Return, iNumber, iWidth)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_retnllen()

Returns a long number, with specific width

Syntax

C Prototype

```
#include <hbapi.h>
hb_retnllen( long lNumber, int iWidth ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_retnllen(lNumber, iWidth) --> hb_itemPutNLLen(&hb_stack.Return, lNumber, iWidth)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_reta()

Returns an array with a specific length

Syntax

C Prototype

```
#include <hbapi.h>
hb_reta( ULONG ulLen ) --> void
```

Arguments

Returns

Description

Note that when HB_API_MACROS is defined, this function is replaced with a macro: hb_reta(ulLen) --> hb_arrayNew(&hb_stack.Return, ulLen)

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_storc()

Stores a szString on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_storc( char * szText, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_storclen()

Stores a fixed length string on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_storclen( char * szText, ULONG ulLength, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stords()

SzDate must have yyyymmdd format

Syntax

C Prototype

```
#include <hbapi.h>
hb_stords( char * szDate, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_storl()

Stores a logical integer on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_storl( int iLogical, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_storni()

Stores an integer on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_storni( int iValue, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stornl()

Stores a long on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_stornl( long lValue, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stornd()

Stores a double on a variable by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_stornd( double dValue, int iParam, ... ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xinit()

Initialize fixed memory subsystem

Syntax

C Prototype

```
#include <hbapi.h>
hb_xinit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xexit()

Deinitialize fixed memory subsystem

Syntax

C Prototype

```
#include <hbapi.h>
hb_xexit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xalloc()

Allocates memory, returns NULL on failure

Syntax

C Prototype

```
#include <hbapi.h>
hb_xalloc( ULONG ulSize ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xgrab()

Allocates memory, exits on failure

Syntax

C Prototype

```
#include <hbapi.h>
hb_xgrab( ULONG ulSize ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xfree()

Frees memory

Syntax

C Prototype

```
#include <hbapi.h>
hb_xfree( void * pMem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xrealloc()

Reallocates memory

Syntax

C Prototype

```
#include <hbapi.h>
hb_xrealloc( void * pMem, ULONG ulSize ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xsize()

Returns the size of an allocated memory block

Syntax

C Prototype

```
#include <hbapi.h>
hb_xsize( void * pMem ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xquery()

Query different types of memory information

Syntax

C Prototype

```
#include <hbapi.h>
hb_xquery( USHORT uiMode ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xmemcpy()

Copy more than memcpy() can

Syntax

C Prototype

```
#include <hbapi.h>
hb_xmemcpy( void * pDestArg, void * pSourceArg, ULONG ulLen ) --> ( void * )pResult
```

Arguments

Returns

Description

If UINT_MAX is defined as ULONG_MAX then this function is replaced by a macro replacement to memcpy()

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_xmemset()

Set more than memset() can

Syntax

C Prototype

```
#include <hbapi.h>
hb_xmemset( void * pDestArg, int iFill, ULONG ulLen ) --> ( void * )pResult
```

Arguments

Returns

Description

If UINT_MAX is defined as ULONG_MAX then this function is replaced by a macro replacement to memset()

Examples

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayNew()

Creates a new array

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayNew( PHB_ITEM pItem, ULONG ulLen ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayLen()

Retrives the array len

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayLen( PHB_ITEM pArray ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayIsObject()

Retrives if the array is an object

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayIsObject( PHB_ITEM pArray ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayAdd()

Add a new item to the end of an array item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayAdd( PHB_ITEM pArray, PHB_ITEM pItemValue ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayIns()

Insert a nil item into an array, without changing the length

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayIns( PHB_ITEM pArray, ULONG ulIndex ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayDel()

Delete an array item, without changing length

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayDel( PHB_ITEM pArray, ULONG ulIndex ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arraySize()

Sets the array total length

Syntax

C Prototype

```
#include <hbapi.h>
hb_arraySize( PHB_ITEM pArray, ULONG ulLen ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayLast()

Retrieve last item in an array

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayLast( PHB_ITEM pArray, PHB_ITEM pResult ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayRelease()

Releases an array - don't call it - use ItemRelease() !!!

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayRelease( PHB_ITEM pArray ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arraySet()

Sets an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arraySet( PHB_ITEM pArray, ULONG ulIndex, PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGet()

Retrieves an item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGet( PHB_ITEM pArray, ULONG ulIndex, PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetItemPtr()

Returns pointer to specified element of the array

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetItemPtr( PHB_ITEM pArray, ULONG ulIndex ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayCopyC()

Copy a string into an array item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayCopyC( PHB_ITEM pArray, ULONG ulIndex, char * szBuffer, ULONG ulLen ) --> (
ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetC()

Retrieves the string contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetC( PHB_ITEM pArray, ULONG ulIndex ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetCPtr()

Retrieves the string pointer on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetCPtr( PHB_ITEM pArray, ULONG ulIndex ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetCLen()

Retrieves the string length contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetCLen( PHB_ITEM pArray, ULONG ulIndex ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetL()

Retrieves the logical value contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetL( PHB_ITEM pArray, ULONG ulIndex ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetNI()

Retrieves the int value contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetNI( PHB_ITEM pArray, ULONG ulIndex ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetNL()

Retrieves the long numeric value contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetNL( PHB_ITEM pArray, ULONG ulIndex ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetND()

Retrieves the double value contained on an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetND( PHB_ITEM pArray, ULONG ulIndex ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetDS()

Retrieves the date value contained in an array element

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetDS( PHB_ITEM pArray, ULONG ulIndex, char * szDate ) --> ( char *
)pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetDL()

Retrieves the date value contained in an array element, as a long integer

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetDL( PHB_ITEM pArray, ULONG ulIndex ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayGetType()

Retrieves the type of an array item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayGetType( PHB_ITEM pArray, ULONG ulIndex ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayFill()

Fill an array with a given item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayFill( PHB_ITEM pArray, PHB_ITEM pValue, ULONG * pulStart, ULONG * pulCount )
--> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayScan()

Scan an array for a given item, or until code-block item returns TRUE

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayScan( PHB_ITEM pArray, PHB_ITEM pValue, ULONG * pulStart, ULONG * pulCount )
--> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayEval()

Execute a code-block for every element of an array item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayEval( PHB_ITEM pArray, PHB_ITEM bBlock, ULONG * pulStart, ULONG * pulCount )
--> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayCopy()

Copy items from one array to another

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayCopy( PHB_ITEM pSrcArray, PHB_ITEM pDstArray, ULONG * pulStart, ULONG *
pulCount, ULONG * pulTarget ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayClone()

Returns a duplicate of an existing array, including all nested items

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayClone( PHB_ITEM pArray ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arraySort()

Sorts an array item

Syntax

C Prototype

```
#include <hbapi.h>
hb_arraySort( PHB_ITEM pArray, ULONG * pulStart, ULONG * pulCount, PHB_ITEM pBlock )
--> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stricmp()

Compare two strings without regards to case

Syntax

C Prototype

```
#include <hbapi.h>
hb_stricmp( const char * s1, const char * s2 ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strnicmp()

Compare two string without regards to case, limited by length

Syntax

C Prototype

```
#include <hbapi.h>
hb_strnicmp( const char * s1, const char * s2, ULONG ulLen ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strupr()

Convert a string in-place to upper-case

Syntax

C Prototype

```
#include <hbapi.h>
hb_strupr( char * pszText ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strdup()

Returns a pointer to a newly allocated copy of the source string

Syntax

```
C Prototype

#include <hbapi.h>
hb_strdup( const char * pszText ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strMatchRegExp()

Compare two strings using a regular expression pattern

Syntax

C Prototype

```
#include <hbapi.h>
hb_strMatchRegExp( const char * szString, const char * szMask ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strEmpty()

Returns whether a string contains only white space

Syntax

C Prototype

```
#include <hbapi.h>
hb_strEmpty( const char * szText, ULONG ulLen ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strDescend()

Copy a string to a buffer, inverting each character

Syntax

C Prototype

```
#include <hbapi.h>
hb_strDescend( char * szStringTo, const char * szStringFrom, ULONG ulLen ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strAt()

Returns an index to a sub-string within another string

Syntax

C Prototype

```
#include <hbapi.h>
hb_strAt( const char * szSub, ULONG ulSubLen, const char * szText, ULONG ulLen ) -->
( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strUpper()

Convert an existing string buffer to upper case

Syntax

C Prototype

```
#include <hbapi.h>
hb_strUpper( char * szText, ULONG ulLen ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strLower()

Convert an existing string buffer to lower case

Syntax

C Prototype

```
#include <hbapi.h>
hb_strLower( char * szText, ULONG ulLen ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strncpyUpper()

Copy an existing string buffer to another buffer, as upper case

Syntax

C Prototype

```
#include <hbapi.h>
hb_strncpyUpper( char * pDest, const char * pSource, ULONG ulLen ) --> ( char *
)pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strVal()

Return the numeric value of a character string representation of a number

Syntax

C Prototype

```
#include <hbapi.h>
hb_strVal( const char * szText, ULONG ulLen ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strLTrim()

Return a pointer to the first non-white space character

Syntax

C Prototype

```
#include <hbapi.h>
hb_strLTrim( const char * szText, ULONG * ulLen ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_strRTrimLen()

Return length of a string, ignoring trailing white space (or true spaces)

Syntax

C Prototype

```
#include <hbapi.h>
hb_strRTrimLen( const char * szText, ULONG ulLen, BOOL bAnySpace ) --> ( ULONG
)ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_numRound()

Round a number to a specific number of digits

Syntax

C Prototype

```
#include <hbapi.h>
hb_numRound( double dResult, int iDec ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_clsReleaseAll()

Releases all defined classes

Syntax

C Prototype

```
#include <hbapi.h>
hb_clsReleaseAll( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_objGetClsName()

Retrieves an object class name

Syntax

C Prototype

```
#include <hbapi.h>
hb_objGetClsName( PHB_ITEM pObject ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_objGetMethod()

Returns the method pointer of a object class

Syntax

C Prototype

```
#include <hbapi.h>
hb_objGetMethod( PHB_ITEM pObject, PHB_SYMB pSymMsg ) --> ( PHB_FUNC )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_objHasMsg()

Returns TRUE/FALSE whether szString is an existing message for object

Syntax

C Prototype

```
#include <hbapi.h>
hb_objHasMsg( PHB_ITEM pObject, char * szString ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymGet()

Finds and creates a dynamic symbol if not found

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymGet( char * szName ) --> ( PHB_DYNS )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymNew()

Creates a new dynamic symbol based on a local one

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymNew( PHB_SYMB pSymbol ) --> ( PHB_DYNS )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymFind()

Finds a dynamic symbol

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymFind( char * szName ) --> ( PHB_DYNS )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymFindName()

Converts to uppercase and finds a dynamic symbol

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymFindName( char * szName ) --> ( PHB_DYNS )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymLog()

Displays all dynamic symbols

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymLog( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymRelease()

Releases the memory of the dynamic symbol table

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymRelease( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_dynsymEval()

Enumerates all dynamic symbols

Syntax

C Prototype

```
#include <hbapi.h>
hb_dynsymEval( PHB_DYNS_FUNC pFunction, void * Cargo ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargInit()

Initialize command line argument API's

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargInit( int argc, char * argv[] ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargARGC()

Retrieve command line argument count

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargARGC( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargARGV()

Retrieve command line argument buffer pointer

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargARGV( void ) --> ( char ** )ppszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargIsInternal()

Determine if a string is an internal setting

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargIsInternal( const char * szArg ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargCheck()

Check if a given internal switch (like //INFO) was set

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargCheck( const char * pszName ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargString()

Returns the string value of an internal switch (like //TEMPPATH:"C:\")

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargString( const char * pszName ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargNum()

Returns the numeric value of an internal switch (like //F:90)

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargNum( const char * pszName ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_cmdargProcessVM()

Check for command line internal arguments

Syntax

C Prototype

```
#include <hbapi.h>
hb_cmdargProcessVM( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_symbolNew()

Create a new symbol

Syntax

C Prototype

```
#include <hbapi.h>
hb_symbolNew( char * szName ) --> ( PHB_SYMB )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockNew()

Create a code-block

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockNew( BYTE * pBuffer, USHORT uiLocals, USHORT * pLocalPosTable, PHB_SYMB
pSymbols ) --> ( HB_CODEBLOCK_PTR )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockMacroNew()

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockMacroNew( BYTE * pBuffer, USHORT usLen ) --> ( HB_CODEBLOCK_PTR )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockDelete()

Delete a codeblock

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockDelete( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockGetVar()

Get local variable referenced in a codeblock

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockGetVar( PHB_ITEM pItem, LONG iItemPos ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockGetRef()

Get local variable passed by reference

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockGetRef( PHB_ITEM pItem, PHB_ITEM pRefer ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockEvaluate()

Evaluate a codeblock

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockEvaluate( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockCopy()

Copy a codeblock

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockCopy( PHB_ITEM pDest, PHB_ITEM pSource ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarValueNew()

Create a new global value

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarValueNew( HB_ITEM_PTR pSource, BOOL bTrueMemvar ) --> ( HB_HANDLE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarValueBaseAddress()

Retrieve the base address of the values table

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarValueBaseAddress( void ) --> ( HB_VALUE_PTR * )phResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarsInit()

Initialize the memvar API system

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarsInit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarsRelease()

Clear all PUBLIC and PRIVATE variables

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarsRelease( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarsFree()

Release the memvar API system

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarsFree( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarValueIncRef()

Increase the reference count of a global value

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarValueIncRef( HB_HANDLE hValue ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarValueDecRef()

Decrease the reference count of a global value

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarValueDecRef( HB_HANDLE hValue ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarSetValue()

Copy an item into a symbol

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarSetValue( PHB_SYMB pMemvarSymb, HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarGet()

Copy an symbol value into an item

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarGet( HB_ITEM_PTR pItem, PHB_SYMB pMemvarSymb ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarGetValue()

Copy an symbol value into an item, with error trapping

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarGetValue( HB_ITEM_PTR pItem, PHB_SYMB pMemvarSymb ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarGetRefer()

Copy a reference to a symbol value into an item, with error trapping

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarGetRefer( HB_ITEM_PTR pItem, PHB_SYMB pMemvarSymb ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarGetPrivatesBase()

Retrieve current PRIVATE variables stack base

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarGetPrivatesBase( void ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarSetPrivatesBase()

Release PRIVATE variables created after specified base

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarSetPrivatesBase( ULONG ulBase ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarNewParameter()

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarNewParameter( PHB_SYMB pSymbol, PHB_ITEM pValue ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarGetStrValuePtr()

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarGetStrValuePtr( char * szVarName, ULONG *pulLen ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarCreateFromItem()

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarCreateFromItem( PHB_ITEM pMemvar, BYTE bScope, PHB_ITEM pValue ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarScope()

Retrieve scope of a dynamic variable symbol

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarScope( char * szVarName, ULONG ulLength ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conInit()

Initialize the console API system

Syntax

C Prototype

```
#include <hbapi.h>
hb_conInit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conRelease()

Release the console API system

Syntax

C Prototype

```
#include <hbapi.h>
hb_conRelease( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conNewLine()

Retrieve a pointer to a static buffer containing new-line characters

Syntax

C Prototype

```
#include <hbapi.h>
hb_conNewLine( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conOutStd()

Output an string to STDOUT

Syntax

C Prototype

```
#include <hbapi.h>
hb_conOutStd( char * pStr, ULONG ulLen ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conOutErr()

Output an string to STDERR

Syntax

C Prototype

```
#include <hbapi.h>
hb_conOutErr( char * pStr, ULONG ulLen ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conSetCursor()

Retrieve and optionally set cursor shape

Syntax

C Prototype

```
#include <hbapi.h>
hb_conSetCursor( BOOL bSetCursor, USHORT usNewCursor ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conSetColor()

Retrieve and optionally set console color

Syntax

C Prototype

```
#include <hbapi.h>
hb_conSetColor( char * szColor ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_conXSaveRestRelease()

Release the save/restore API

Syntax

C Prototype

```
#include <hbapi.h>
hb_conXSaveRestRelease( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_compReservedName()

Determines if a string contains a reserve word

Syntax

C Prototype

```
#include <hbapi.h>
hb_compReservedName( char * szName ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_procname()

Retrieve a procedure name into a buffer

Syntax

C Prototype

```
#include <hbapi.h>
hb_procname( int iLevel, char * szName ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroGetValue()

Retrieve results of a macro expansion

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroGetValue( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroSetValue()

Assign a value to a macro-expression item

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroSetValue( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroTextValue()

Macro text substitution

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroTextValue( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroPushSymbol()

Handle a macro function calls, e.g. var := &macro()

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroPushSymbol( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroRun()

Executes pcode compiled by macro compiler

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroRun( HB_MACRO_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroCompile()

Compile a string and return a pcode buffer

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroCompile( char * szString ) --> ( HB_MACRO_PTR )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroDelete()

Release all memory allocated for macro evaluation

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroDelete( HB_MACRO_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroTextSubst()

Substitute macro variables occurences within a given string

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroTextSubst( char * szString, ULONG *pulStringLen ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroIsIdent()

Determine if a string is a valid function or variable name

Syntax

```
C Prototype

#include <hbapi.h>
hb_macroIsIdent( char * szString ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroPopAliasedValue()

Compiles and evaluates an aliased macro expression

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroPopAliasedValue( HB_ITEM_PTR pAlias, HB_ITEM_PTR pVar ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroPushAliasedValue()

Compiles and evaluates an aliased macro expression

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroPushAliasedValue( HB_ITEM_PTR pAlias, HB_ITEM_PTR pVar ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_macroGetType()

Determine the type of an expression

Syntax

C Prototype

```
#include <hbapi.h>
hb_macroGetType( HB_ITEM_PTR pItem ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcAlloc()

Allocates a memory controlled by the garbage collector

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcAlloc( ULONG ulSize, HB_GARBAGE_FUNC_PTR pFunc ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcFree()

Deallocates a memory allocated by the garbage collector

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcFree( void *pAlloc ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcLock()

Do not release passed memory block

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcLock( void *pAlloc ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcUnlock()

Passed block is allowed to be released

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcUnlock( void *pAlloc ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcLockItem()

Do not release a memory block stored inside an item

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcLockItem( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcUnlockItem()

Allow to release the item

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcUnlockItem( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcCollect()

Checks if a single memory block can be released

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcCollect( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcCollectAll()

Checks if all memory blocks can be released

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcCollectAll( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_gcItemRef()

Checks if passed item refers passed memory block pointer

Syntax

C Prototype

```
#include <hbapi.h>
hb_gcItemRef( HB_ITEM_PTR pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmIsLocalRef()

Hvm.c - mark all local variables as used

Syntax

C Prototype

```
#include <hbapi.h>
hb_vmIsLocalRef( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmIsStaticRef()

Hvm.c - mark all static variables as used

Syntax

C Prototype

```
#include <hbapi.h>
hb_vmIsStaticRef( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_memvarsIsMemvarRef()

Memvars.c - mark all memvar variables as used

Syntax

C Prototype

```
#include <hbapi.h>
hb_memvarsIsMemvarRef( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_clsIsClassRef()

Classes.c - mark all class internals as used

Syntax

C Prototype

```
#include <hbapi.h>
hb_clsIsClassRef( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_codeblockDeleteGarbage()

Clear a codeblock before releasing by the GC

Syntax

C Prototype

```
#include <hbapi.h>
hb_codeblockDeleteGarbage( void * Cargo ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_arrayReleaseGarbage()

Clear an array before releasing by the GC

Syntax

C Prototype

```
#include <hbapi.h>
hb_arrayReleaseGarbage( void * Cargo ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_idleState()

Services a single idle state

Syntax

C Prototype

```
#include <hbapi.h>
hb_idleState( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_idleReset()

Services a single idle state

Syntax

C Prototype

```
#include <hbapi.h>
hb_idleReset( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_idleShutDown()

Closes all background tasks

Syntax

C Prototype

```
#include <hbapi.h>
hb_idleShutDown( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_verPlatform()

Retrieves a newly allocated buffer containing platform version

Syntax

C Prototype

```
#include <hbapi.h>
hb_verPlatform( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_verCompiler()

Retrieves a newly allocated buffer containing compiler version

Syntax

C Prototype

```
#include <hbapi.h>
hb_verCompiler( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_verHarbour()

Retrieves a newly allocated buffer containing harbour version

Syntax

C Prototype

```
#include <hbapi.h>
hb_verHarbour( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_verBuildInfo()

Display harbour, compiler, and platform versions to standard console

Syntax

C Prototype

```
#include <hbapi.h>
hb_verBuildInfo( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

HB_IS_OF_TYPE()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_OF_TYPE( p, t ) --> <see ( ( ( p )->type & ~HB_IT_BYREF ) == t )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_BYREF()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_BYREF( p ) --> <see ( ( p )->type & HB_IT_BYREF )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_ARRAY()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_ARRAY( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_ARRAY )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_NIL()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_NIL( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_NIL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_BLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_BLOCK( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_BLOCK )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_DATE()

Syntax

```
C Prototype (macro definition)

#include <hbapi.h>
HB_IS_DATE( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_DATE )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_DOUBLE()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_DOUBLE( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_DOUBLE )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_INTEGER()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_INTEGER( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_INTEGER )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_LOGICAL()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_LOGICAL( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_LOGICAL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_LONG()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_LONG( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_LONG )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_NUMERIC()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_NUMERIC( p ) --> <see ( ( p )->type & HB_IT_NUMERIC )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_OBJECT()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_OBJECT( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_OBJECT )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_STRING()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_STRING( p ) --> <see ( ( ( p )->type & ~( HB_IT_BYREF | HB_IT_MEMOFLAG ) ) ==
HB_IT_STRING )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_MEMO()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_MEMO( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_MEMO )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_SYMBOL()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_SYMBOL( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_SYMBOL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_MEMVAR()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_MEMVAR( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_MEMVAR )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_IS_POINTER()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_IS_POINTER( p ) --> <see HB_IS_OF_TYPE( p, HB_IT_POINTER )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISNIL()

NOTE: Intentionally using a different method

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISNIL( n ) --> <see ( hb_param( n, HB_IT_ANY ) == NULL || HB_IS_NIL( hb_param( n,
HB_IT_ANY ) ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISCHAR()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISCHAR( n ) --> <see ( hb_param( n, HB_IT_STRING ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISNUM()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISNUM( n ) --> <see ( hb_param( n, HB_IT_NUMERIC ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISLOG()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISLOG( n ) --> <see ( hb_param( n, HB_IT_LOGICAL ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISDATE()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISDATE( n ) --> <see ( hb_param( n, HB_IT_DATE ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISMEMO()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISMEMO( n ) --> <see ( hb_param( n, HB_IT_MEMO ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISBYREF()

NOTE: Intentionally using a different method

Syntax

```
C Prototype (macro definition)

#include <hbapi.h>
ISBYREF( n ) --> <see ( hb_parinfo( n ) & HB_IT_BYREF )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISARRAY()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISARRAY( n ) --> <see ( hb_param( n, HB_IT_ARRAY ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISOBJECT()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISOBJECT( n ) --> <see ( ISARRAY( n ) && hb_param( n, HB_IT_ARRAY
)->asArray.value->uiClass != 0 )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISBLOCK()

Not available in CA-Cl*pper.

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
ISBLOCK( n ) --> <see ( hb_param( n, HB_IT_BLOCK ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

ISPOINTER()

Not available in CA-Cl*pper.

Syntax

```
C Prototype (macro definition)

#include <hbapi.h>
ISPOINTER( n ) --> <see ( hb_param( n, HB_IT_POINTER ) != NULL )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

HB_ISSPACE()

Syntax

C Prototype (macro definition)

```
#include <hbapi.h>
HB_ISSPACE( c ) --> <see ( ( c ) == ' ' || ( c ) == HB_CHAR_HT || ( c ) ==
HB_CHAR_LF || ( c ) == HB_CHAR_CR )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapi.h

Platforms

All

hb_errGetDescription()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetDescription( PHB_ITEM pError ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetFileName()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetFileName( PHB_ITEM pError ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetFlags()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetFlags( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetGenCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetGenCode( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetOperation()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetOperation( PHB_ITEM pError ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetOsCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetOsCode( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetSeverity()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetSeverity( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetSubCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetSubCode( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetSubSystem()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetSubSystem( PHB_ITEM pError ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errGetTries()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errGetTries( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errLaunch()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errLaunch( PHB_ITEM pError ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errNew()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errNew( void ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutArgs()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutArgs( PHB_ITEM pError, USHORT uiArgCount, ... ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutDescription()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutDescription( PHB_ITEM pError, char * szDescription ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutFileName()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutFileName( PHB_ITEM pError, char * szFileName ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutFlags()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutFlags( PHB_ITEM pError, USHORT uiFlags ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutGenCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutGenCode( PHB_ITEM pError, USHORT uiGenCode ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutOperation()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutOperation( PHB_ITEM pError, char * szOperation ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutOsCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutOsCode( PHB_ITEM pError, USHORT uiOsCode ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutSeverity()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutSeverity( PHB_ITEM pError, USHORT uiSeverity ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutSubCode()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutSubCode( PHB_ITEM pError, USHORT uiSubCode ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutSubSystem()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutSubSystem( PHB_ITEM pError, char * szSubSystem ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errPutTries()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errPutTries( PHB_ITEM pError, USHORT uiTries ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRelease()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRelease( PHB_ITEM pError ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errInit()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errInit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errExit()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errExit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errLaunchSubst()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errLaunchSubst( PHB_ITEM pError ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_New()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_New( USHORT uiSeverity, char * szSubSystem, ULONG ulGenCode, ULONG
ulSubCode, char * szDescription, char * szOperation, USHORT uiOsCode, USHORT
uiFlags ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_New_Subst()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_New_Subst( USHORT uiSeverity, char * szSubSystem, ULONG ulGenCode, ULONG
ulSubCode, char * szDescription, char * szOperation, USHORT uiOsCode, USHORT
uiFlags ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_BASE()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_BASE( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_BASE_Ext1()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_BASE_Ext1( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation, USHORT uiOsCode, USHORT uiFlags ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_BASE_Subst()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_BASE_Subst( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_BASE_SubstR()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_BASE_SubstR( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_TERM()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_TERM( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation, USHORT uiOSCode, USHORT uiFlags ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_DBCMD()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_DBCMD( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errRT_TOOLS()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errRT_TOOLS( ULONG ulGenCode, ULONG ulSubCode, char * szDescription, char *
szOperation ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errInternal()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errInternal( ULONG ulIntCode, char * szText, char * szPar1, char * szPar2 ) -->
void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_errorHandler()

Syntax

C Prototype

```
#include <hbapierr.h>
hb_errorHandler( HB_ERROR_INFO_PTR pNewHandler ) --> ( HB_ERROR_INFO_PTR )hParam
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsChDir()

Change working directory

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsChDir( BYTE * pszDirName ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsChDrv()

Change working drive

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsChDrv( BYTE nDrive ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsClose()

Close a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsClose( FHANDLE hFileHandle ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCommit()

Commit updates of a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCommit( FHANDLE hFileHandle ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCreate()

Create a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCreate( BYTE * pszFileName, USHORT uiAttribute ) --> ( FHANDLE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCreateTemp()

Create a temporary file from components

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCreateTemp( const BYTE * pszDir, const BYTE * pszPrefix, USHORT uiAttribute )
--> ( FHANDLE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCurDir()

Retrieve a static pointer containing current directory for specified drive

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCurDir( USHORT uiDrive ) --> ( BYTE * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCurDirBuff()

Copy current directory for given drive into a buffer

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCurDirBuff( USHORT uiDrive, BYTE * pbyBuffer, ULONG ulLen ) --> ( USHORT
)usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsCurDrv()

Retrieve current drive number

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsCurDrv( void ) --> ( BYTE )cResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsDelete()

Delete a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsDelete( BYTE * pszFileName ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsEof()

Determine if an open file is position at end-of-file

Syntax

```
C Prototype

#include <hbapifs.h>
hb_fsEof( FHANDLE hFileHandle ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsError()

Retrieve file system error

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsError( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsFile()

Determine if a file exists

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsFile( BYTE * pszFileName ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsFSize()

Determine the size of a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsFSize( BYTE * pszFileName, BOOL bUseDirEntry ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsExtOpen()

Open a file using default extension and a list of paths

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsExtOpen( BYTE * pszFileName, BYTE * pDefExt, USHORT uiFlags, BYTE * pPaths,
PHB_ITEM pError ) --> ( FHANDLE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsIsDrv()

Determine if a drive number is a valid drive

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsIsDrv( BYTE nDrive ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsIsDevice()

Determine if a file is attached to a device (console?)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsIsDevice( FHANDLE hFileHandle ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsLock()

Request a lock on a portion of a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsLock( FHANDLE hFileHandle, ULONG ulStart, ULONG ulLength, USHORT uiMode ) --> (
BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsMkDir()

Create a directory

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsMkDir( BYTE * pszDirName ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsOpen()

Open a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsOpen( BYTE * pszFileName, USHORT uiFlags ) --> ( FHANDLE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsRead()

Read contents of a file into a buffer (<=64K)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsRead( FHANDLE hFileHandle, BYTE * pBuff, USHORT ulCount ) --> ( USHORT
)usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsReadLarge()

Read contents of a file into a buffer (>64K)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsReadLarge( FHANDLE hFileHandle, BYTE * pBuff, ULONG ulCount ) --> ( ULONG
)ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsRmDir()

Remove a directory

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsRmDir( BYTE * pszDirName ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsRename()

Rename a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsRename( BYTE * pszOldName, BYTE * pszNewName ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsSeek()

Reposition an open file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsSeek( FHANDLE hFileHandle, LONG lOffset, USHORT uiMode ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsTell()

Retrieve the current position of a file

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsTell( FHANDLE hFileHandle ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsTempName()

Create a temporary file name in a buffer

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsTempName( BYTE * pszBuffer, const BYTE * pszDir, const BYTE * pszPrefix ) -->
void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsSetDevMode()

Change the device mode of a file (text/binary)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsSetDevMode( FHANDLE hFileHandle, USHORT uiDevMode ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsSetDevRaw()

Change the device mode of a file to raw (binary)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsSetDevRaw( FHANDLE hFileHandle ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsSetDevText()

Change the device mode of a file to text

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsSetDevText( FHANDLE hFileHandle ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsSetError()

Set the file system error number

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsSetError( USHORT uiError ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsWrite()

Write to an open file from a buffer (<=64K)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsWrite( FHANDLE hFileHandle, BYTE * pBuff, USHORT ulCount ) --> ( USHORT
)usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsWriteLarge()

Write to an open file from a buffer (>64K)

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsWriteLarge( FHANDLE hFileHandle, BYTE * pBuff, ULONG ulCount ) --> ( ULONG
)ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsFNameSplit()

Split given filename into path, name and extension

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsFNameSplit( char * pszFileName ) --> ( PHB_FNAME )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsFNameMerge()

This function joins path, name and extension into a string with a filename

Syntax

C Prototype

```
#include <hbapifs.h>
hb_fsFNameMerge( char * pszFileName, PHB_FNAME pFileName ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_fsFLock()

Syntax

C Prototype (macro definition)

```
#include <hbapifs.h>
hb_fsFLock( h, s, l ) --> <see hb_fsLock( h, s, l, FL_LOCK )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapifs.h

Platforms

All

hb_fsFUnlock()

Syntax

C Prototype (macro definition)

```
#include <hbapifs.h>
hb_fsFUnlock( h, s, l ) --> <see hb_fsLock( h, s, l, FL_UNLOCK )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapifs.h

Platforms

All

hb_gtInit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtInit( int iFilenoStdin, int iFilenoStdout, int iFilenoStderr ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtExit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtExit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtAdjustPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtAdjustPos( int iHandle, char * pStr, ULONG ulLen ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtBox()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtBox( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbyFrame ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtBoxD()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtBoxD( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight ) --> (
USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtBoxS()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtBoxS( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight ) --> (
USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtColorSelect()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtColorSelect( USHORT uiColorIndex ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtColorToN()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtColorToN( char * szColorString ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtDispBegin()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtDispBegin( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtDispCount()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtDispCount( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtDispEnd()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtDispEnd( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtDrawShadow()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtDrawShadow( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE
byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtGetBlink()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtGetBlink( BOOL * pbBlink ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtGetColorStr()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtGetColorStr( char * pszColorString ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtGetCursor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtGetCursor( USHORT * puiCursorShape ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtGetPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtGetPos( SHORT * piRow, SHORT * piCol ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtIsColor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtIsColor( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtMaxCol()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtMaxCol( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtMaxRow()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtMaxRow( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtPostExt()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtPostExt( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtPreExt()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtPreExt( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSuspend()

Prepare the reminal for shell output

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSuspend( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtResume()

Resume the terminal after the shell output

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtResume( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtReadKey()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtReadKey( HB_inkey_enum eventmask ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtRectSize()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtRectSize( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, USHORT
* puiBuffSize ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtRepChar()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtRepChar( USHORT uiRow, USHORT uiCol, BYTE byChar, USHORT uiCount ) --> ( USHORT
)usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtRest()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtRest( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, void *
pScrBuff ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSave()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSave( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, void *
pScrBuff ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtScrDim()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtScrDim( USHORT * puiHeight, USHORT * puiWidth ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtScroll()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtScroll( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, SHORT
iRows, SHORT iCols ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetBlink()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetBlink( BOOL bBlink ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetColorStr()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetColorStr( char * pszColorString ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetCursor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetCursor( USHORT uiCursorShape ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetMode()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetMode( USHORT uiRows, USHORT uiCols ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetPos( SHORT iRow, SHORT iCol ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetPosContext()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetPosContext( SHORT iRow, SHORT iCol, SHORT iMode ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetSnowFlag()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetSnowFlag( BOOL bNoSnow ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtTone()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtTone( double dFrequency, double dDuration ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWrite()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWrite( BYTE * pbyStr, ULONG ulLen ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWriteAt()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWriteAt( USHORT uiRow, USHORT uiCol, BYTE * pbyStr, ULONG ulLen ) --> ( USHORT
)usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWriteCon()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWriteCon( BYTE * pbyStr, ULONG ulLen ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtVersion()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtVersion( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWCreate()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWCreate( HB_GT_RECT * rect, HB_GT_WND ** wnd ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWDestroy()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWDestroy( HB_GT_WND * wnd ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWFlash()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWFlash( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWApp()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWApp( HB_GT_WND ** wnd ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWCurrent()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWCurrent( HB_GT_WND * wnd ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWPos( HB_GT_WND * wnd, HB_GT_RECT * rect ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtWVis()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtWVis( HB_GT_WND * wnd, USHORT uiStatus ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSLR()

System Level Request

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSLR( HB_GT_SLR * pSLR ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtModalRead()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtModalRead( void * ) --> ( USHORT )usResult
```

Arguments

<void *>

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtBeginWrite()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtBeginWrite( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtEndWrite()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtEndWrite( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtFlushCursor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtFlushCursor( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetColor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetColor( HB_GT_RGB * color ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtGetColor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtGetColor( HB_GT_RGB * color ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gtSetBorder()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gtSetBorder( HB_GT_RGB * color ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Init()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Init( int iFilenoStdin, int iFilenoStdout, int iFilenoStderr ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Exit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Exit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_AdjustPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_AdjustPos( BYTE * pStr, ULONG ulLen ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Box()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Box( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbyFrame, BYTE byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_BoxD()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_BoxD( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbyFrame, BYTE byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_BoxS()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_BoxS( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbyFrame, BYTE byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Col()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Col( void ) --> ( SHORT )sResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_DispBegin()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_DispBegin( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_DispCount()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_DispCount( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_DispEnd()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_DispEnd( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_GetBlink()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_GetBlink( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_GetCursorStyle()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_GetCursorStyle( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_GetScreenHeight()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_GetScreenHeight( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_GetScreenWidth()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_GetScreenWidth( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_GetText()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_GetText( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbyDst ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_HorizLine()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_HorizLine( USHORT uiRow, USHORT uiLeft, USHORT uiRight, BYTE byChar, BYTE
byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_IsColor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_IsColor( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_PreExt()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_PreExt( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_PostExt()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_PostExt( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Suspend()

Suspend the terminal before the shell call

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Suspend( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Resume()

Resume the terminal after the shell call

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Resume( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Puts()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Puts( USHORT uiRow, USHORT uiCol, BYTE byAttr, BYTE * pbyStr, ULONG ulLen )
--> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_PutText()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_PutText( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE *
pbySrc ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_ReadKey()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_ReadKey( HB_inkey_enum eventmask ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_RectSize()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_RectSize( USHORT rows, USHORT cols ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Replicate()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Replicate( USHORT uiTop, USHORT uiLeft, BYTE byAttr, BYTE byChar, ULONG ulLen
) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Row()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Row( void ) --> ( SHORT )sResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Scroll()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Scroll( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight, BYTE
byAttr, SHORT iRows, SHORT iCols ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_SetAttribute()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_SetAttribute( USHORT uiTop, USHORT uiLeft, USHORT uiBottom, USHORT uiRight,
BYTE byAttr ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_SetBlink()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_SetBlink( BOOL bBlink ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_SetCursorStyle()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_SetCursorStyle( USHORT uiCursorShape ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_SetMode()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_SetMode( USHORT uiRows, USHORT uiCols ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_SetPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_SetPos( SHORT iRow, SHORT iCol, SHORT iMethod ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Tone()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Tone( double dFrequency, double dDuration ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_Version()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_Version( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_gt_VertLine()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_gt_VertLine( USHORT uiCol, USHORT uiTop, USHORT uiBottom, BYTE byChar, BYTE
byAttr ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkey()

Wait for keyboard input

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkey( BOOL bWait, double dSeconds, HB_inkey_enum event_mask ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyGet()

Extract the next key from the Harbour keyboard buffer

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyGet( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyPut()

Inserts an inkey code into the keyboard buffer

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyPut( int ch ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyLast()

Return the value of the last key that was extracted

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyLast( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyNext()

Return the next key without extracting it

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyNext( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyPoll()

Poll the console keyboard to stuff the Harbour buffer

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyPoll( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_inkeyReset()

Reset the Harbour keyboard buffer

Syntax

C Prototype

```
#include <hbapigt.h>
hb_inkeyReset( BOOL allocate ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseIsPresent()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseIsPresent( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseGetCursor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseGetCursor( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseSetCursor()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseSetCursor( BOOL bVisible ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseCol()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseCol( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseRow()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseRow( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseSetPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseSetPos( int iRow, int iCol ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseIsButtonPressed()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseIsButtonPressed( int iButton ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseCountButton()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseCountButton( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseSetBounds()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseSetBounds( int iTop, int iLeft, int iBottom, int iRight ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouseGetBounds()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouseGetBounds( int * piTop, int * piLeft, int * piBottom, int * piRight ) -->
void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Init()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Init( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Exit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Exit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_IsPresent()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_IsPresent( void ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Show()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Show( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Hide()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Hide( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Col()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Col( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_Row()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_Row( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_SetPos()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_SetPos( int iRow, int iCol ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_IsButtonPressed()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_IsButtonPressed( int iButton ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_CountButton()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_CountButton( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_SetBounds()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_SetBounds( int iTop, int iLeft, int iBottom, int iRight ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_mouse_GetBounds()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_mouse_GetBounds( int * piTop, int * piLeft, int * piBottom, int * piRight ) -->
void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_setkeyInit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_setkeyInit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_setkeyExit()

Syntax

C Prototype

```
#include <hbapigt.h>
hb_setkeyExit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is gt* (ie. gtdos)

Platforms

All

hb_evalLaunch()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_evalLaunch( PEVALINFO pEvalInfo ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_evalNew()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_evalNew( PEVALINFO pEvalInfo, PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_evalPutParam()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_evalPutParam( PEVALINFO pEvalInfo, PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_evalRelease()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_evalRelease( PEVALINFO pEvalInfo ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemDo()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemDo( PHB_ITEM pItem, USHORT uiPCount, PHB_ITEM pItemArg1, ... ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemDoC()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemDoC( char * szFunc, USHORT uiPCount, PHB_ITEM pItemArg1, ... ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemArrayGet()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemArrayGet( PHB_ITEM pArray, ULONG ulIndex ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemArrayNew()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemArrayNew( ULONG ulLen ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemArrayPut()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemArrayPut( PHB_ITEM pArray, ULONG ulIndex, PHB_ITEM pItem ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemCopyC()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemCopyC( PHB_ITEM pItem, char * szBuffer, ULONG ulLen ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemFreeC()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemFreeC( char * szText ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetC()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetC( PHB_ITEM pItem ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetCPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetCPtr( PHB_ITEM pItem ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetCLen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetCLen( PHB_ITEM pItem ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetDS()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetDS( PHB_ITEM pItem, char * szDate ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetDL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetDL( PHB_ITEM pItem ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetL( PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetND()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetND( PHB_ITEM pItem ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetNI()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetNI( PHB_ITEM pItem ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetNL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetNL( PHB_ITEM pItem ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetNLen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetNLen( PHB_ITEM pItem, int * piWidth, int * piDec ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemGetPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemGetPtr( PHB_ITEM pItem ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemNew()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemNew( PHB_ITEM pNull ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemInit()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemInit( PHB_ITEM pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPCount()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPCount( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemParam()

Creates a copy of an item parameter (outside the eval stack)

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemParam( USHORT uiParam ) --> ( PHB_ITEM )pResult
```

Arguments

<uiParam> The 1-based parameter to copy and retrieve.

Returns

Description

Use this function whenever the pointer needs to be accessed after the current function returns; for example, if the pointer is to be copied to a static variable or structure member for later access.

Compare to hb_param(), which simply gets a direct pointer to the item on the stack.

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

See Also:

hb_param()

hb_itemPutC()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutC( PHB_ITEM pItem, char * szText ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutCPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutCPtr( PHB_ITEM pItem, char * szText, ULONG ulLen ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutCL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutCL( PHB_ITEM pItem, char * szText, ULONG ulLen ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemSetCMemo()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemSetCMemo( PHB_ITEM pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutD()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutD( PHB_ITEM pItem, long lYear, long lMonth, long lDay ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutDS()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutDS( PHB_ITEM pItem, char * szDate ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutDL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutDL( PHB_ITEM pItem, long lJulian ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutL( PHB_ITEM pItem, BOOL bValue ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutND()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutND( PHB_ITEM pItem, double dNumber ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNI()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNI( PHB_ITEM pItem, int iNumber ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNL()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNL( PHB_ITEM pItem, long lNumber ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNLen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNLen( PHB_ITEM pItem, double dNumber, int iWidth, int iDec ) --> (
PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNDLen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNDLen( PHB_ITEM pItem, double dNumber, int iWidth, int iDec ) --> (
PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNILen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNILen( PHB_ITEM pItem, int iNumber, int iWidth ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutNLLen()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutNLLen( PHB_ITEM pItem, long lNumber, int iWidth ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPutPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPutPtr( PHB_ITEM pItem, void * pValue ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemRelease()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemRelease( PHB_ITEM pItem ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemReturn()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemReturn( PHB_ITEM pItem ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemSize()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemSize( PHB_ITEM pItem ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemType()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemType( PHB_ITEM pItem ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemTypeStr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemTypeStr( PHB_ITEM pItem ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemParamPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemParamPtr( USHORT uiParam, int iMask ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemReturnPtr()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemReturnPtr( void ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemStrCmp()

Our string compare

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemStrCmp( PHB_ITEM pFirst, PHB_ITEM pSecond, BOOL bForceExact ) --> ( int
)iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemCopy()

Copies an item to one place to another respecting its containts

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemCopy( PHB_ITEM pDest, PHB_ITEM pSource ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemClear()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemClear( PHB_ITEM pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemUnRef()

De-references passed variable

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemUnRef( PHB_ITEM pItem ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemStr()

Convert a number to a string

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemStr( PHB_ITEM pNumber, PHB_ITEM pWidth, PHB_ITEM pDec ) --> ( char *
)pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemString()

Convert any scalar to a string

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemString( PHB_ITEM pItem, ULONG * ulLen, BOOL * bFreeReq ) --> ( char *
)pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemValToStr()

Convert any scalar to a string

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemValToStr( PHB_ITEM pItem ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemPadConv()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemPadConv( PHB_ITEM pItem, char * buffer, ULONG * pulSize ) --> ( char *
)pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_itemSwap()

Syntax

C Prototype

```
#include <hbapiitm.h>
hb_itemSwap( PHB_ITEM pItem1, PHB_ITEM pItem2 ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_langRegister()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langRegister( PHB_LANG lang ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langDeRegister()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langDeRegister( char * pszID ) --> ( BOOL )bResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langFind()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langFind( char * pszID ) --> ( PHB_LANG )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langSelect()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langSelect( PHB_LANG lang ) --> ( PHB_LANG )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langSelectID()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langSelectID( char * pszID ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langDGetItem()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langDGetItem( int iIndex ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langID()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langID( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langName()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langName( void ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

hb_langDGetErrorDesc()

Syntax

C Prototype

```
#include <hbapilng.h>
hb_langDGetErrorDesc( ULONG ulIndex ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is lang

Platforms

All

HB_LANG_REQUEST()

Syntax

C Prototype (macro definition)

```
#include <hbapilng.h>
HB_LANG_REQUEST( id ) --> <see HB_LANG_REQUEST_( id )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapilng.h

Platforms

All

hb_rddInsertAreaNode()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddInsertAreaNode( char *szDriver ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddGetCurrentWorkAreaNumber()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddGetCurrentWorkAreaNumber( void ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddGetCurrentWorkAreaPointer()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddGetCurrentWorkAreaPointer( void ) --> ( void * )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddSelectWorkAreaAlias()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddSelectWorkAreaAlias( char * szAlias ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddSelectWorkAreaNumber()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddSelectWorkAreaNumber( int iArea ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddSelectWorkAreaSymbol()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddSelectWorkAreaSymbol( PHB_SYMB pSymAlias ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddGetFieldValue()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddGetFieldValue( HB_ITEM_PTR pItem, PHB_SYMB pFieldSymbol ) --> ( ERRCODE
)hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddPutFieldValue()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddPutFieldValue( HB_ITEM_PTR pItem, PHB_SYMB pFieldSymbol ) --> ( ERRCODE
)hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddFieldGet()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddFieldGet( HB_ITEM_PTR pItem, PHB_SYMB pFieldSymbol ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddFieldPut()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddFieldPut( HB_ITEM_PTR pItem, PHB_SYMB pFieldSymbol ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddShutDown()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddShutDown( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddInherit()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddInherit( PRDDFUNCS pTable, PRDDFUNCS pSubTable, PRDDFUNCS pSuperTable, BYTE *
szDrvName ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddDisinherit()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddDisinherit( BYTE * drvName ) --> ( ERRCODE )hResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddExtendType()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddExtendType( USHORT fieldType ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

hb_rddFieldType()

Syntax

C Prototype

```
#include <hbapirdd.h>
hb_rddFieldType( USHORT extendType ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rdd, nulsys, dbfntx, dbfcdx

Platforms

All

SELF_BOF()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_BOF( w, sp ) --> <see ( ( *( w )->lprfsHost->bof )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_EOF()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_EOF( w, sp ) --> <see ( ( *( w )->lprfsHost->eof )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FOUND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FOUND( w, sp ) --> <see ( ( *( w )->lprfsHost->found )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOTO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOTO( w, l ) --> <see ( ( *( w )->lprfsHost->go )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOTOID()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOTOID( w, sp ) --> <see ( ( *( w )->lprfsHost->goToId )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOBOTTOM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOBOTTOM( w ) --> <see ( ( *( w )->lprfsHost->goBottom )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOTOP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOTOP( w ) --> <see ( ( *( w )->lprfsHost->goTop )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SEEK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SEEK( w, i1, v, i2 ) --> <see ( ( *( w )->lprfsHost->seek )( w, i1, v, i2 ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SKIP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SKIP( w, l ) --> <see ( ( *( w )->lprfsHost->skip )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SKIPFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SKIPFILTER( w, l ) --> <see ( ( *( w )->lprfsHost->skipFilter )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SKIPRAW()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SKIPRAW( w, l ) --> <see ( ( *( w )->lprfsHost->skipRaw )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ADDFIELD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ADDFIELD( w, ip ) --> <see ( ( *( w )->lprfsHost->addField )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_APPEND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_APPEND( w, l ) --> <see ( ( *( w )->lprfsHost->append )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CREATEFIELDS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CREATEFIELDS( w, v ) --> <see ( ( *( w )->lprfsHost->createFields )( w, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_DELETE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_DELETE( w ) --> <see ( ( *( w )->lprfsHost->deleterec )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_DELETED()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_DELETED( w, sp ) --> <see ( ( *( w )->lprfsHost->deleted )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FIELDCOUNT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FIELDCOUNT( w, sp ) --> <see ( ( *( w )->lprfsHost->fieldCount )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FIELDDISPLAY()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FIELDDISPLAY( w, sp ) --> <see ( ( *( w )->lprfsHost->fieldDisplay )( w, sp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FIELDINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FIELDINFO( w, s1, s2, v ) --> <see ( ( *( w )->lprfsHost->fieldInfo )( w, s1,
s2, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FIELDNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FIELDNAME( w, i, bp ) --> <see ( ( *( w )->lprfsHost->fieldName )( w, i, bp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FLUSH()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FLUSH( w ) --> <see ( ( *( w )->lprfsHost->flush )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETREC( w, bpp ) --> <see ( ( *( w )->lprfsHost->getRec )( w, bpp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETVALUE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETVALUE( w, i, v ) --> <see ( ( *( w )->lprfsHost->getValue )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETVARLEN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETVARLEN( w, i, lp ) --> <see ( ( *( w )->lprfsHost->getVarLen )( w, i, lp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOCOLD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOCOLD( w ) --> <see ( ( *( w )->lprfsHost->goCold )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GOHOT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GOHOT( w ) --> <see ( ( *( w )->lprfsHost->goHot )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_PUTVALUE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_PUTVALUE( w, i, v ) --> <see ( ( *( w )->lprfsHost->putValue )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_PUTREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_PUTREC( w, bp ) --> <see ( ( *( w )->lprfsHost->putRec )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RECALL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RECALL( w ) --> <see ( ( *( w )->lprfsHost->recall )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RECCOUNT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RECCOUNT( w, sp ) --> <see ( ( *( w )->lprfsHost->reccount )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RECINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RECINFO( w, v1, i, v2 ) --> <see ( ( *( w )->lprfsHost->recInfo )( w, v1, i, v2
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RECNO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RECNO( w, i ) --> <see ( ( *( w )->lprfsHost->recno )( w, i ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETFIELDEXTENT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETFIELDEXTENT( w, s ) --> <see ( ( *( w )->lprfsHost->setFieldExtent )( w, s )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ALIAS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ALIAS( w, bp ) --> <see ( ( *( w )->lprfsHost->alias )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLOSE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLOSE( w ) --> <see ( ( *( w )->lprfsHost->close )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CREATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CREATE( w, ip ) --> <see ( ( *( w )->lprfsHost->create )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_INFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_INFO( w, i, g ) --> <see ( ( *( w )->lprfsHost->info )( w, i, g ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_NEW()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_NEW( w ) --> <see ( ( *( w )->lprfsHost->newarea )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_OPEN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_OPEN( w, ip ) --> <see ( ( *( w )->lprfsHost->open )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RELEASE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RELEASE( w ) --> <see ( ( *( w )->lprfsHost->release )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_STRUCTSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_STRUCTSIZE( w, sp ) --> <see ( ( *( w )->lprfsHost->structSize )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SYSNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SYSNAME( w, bp ) --> <see ( ( *( w )->lprfsHost->sysName )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_DBEVAL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_DBEVAL( w, ip ) --> <see ( ( *( w )->lprfsHost->dbEval )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_PACK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_PACK( w ) --> <see ( ( *( w )->lprfsHost->pack )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_PACKREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_PACKREC( w, l, sp ) --> <see ( ( *( w )->lprfsHost->packRec )( w, l, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SORT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SORT( w, ip ) --> <see ( ( *( w )->lprfsHost->sort )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_TRANS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_TRANS( w, ip ) --> <see ( ( *( w )->lprfsHost->trans )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_TRANSREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_TRANSREC( w, ip ) --> <see ( ( *( w )->lprfsHost->transRec )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ZAP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ZAP( w ) --> <see ( ( *( w )->lprfsHost->zap )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CHILDEND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CHILDEND( w, ip ) --> <see ( ( *( w )->lprfsHost->childEnd )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CHILDSTART()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CHILDSTART( w, ip ) --> <see ( ( *( w )->lprfsHost->childStart )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CHILDSYNC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CHILDSYNC( w, ip ) --> <see ( ( *( w )->lprfsHost->childSync )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SYNCCHILDREN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SYNCCHILDREN( w ) --> <see ( ( *( w )->lprfsHost->syncChildren )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLEARREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLEARREL( w ) --> <see ( ( *( w )->lprfsHost->clearRel )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FORCEREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FORCEREL( w ) --> <see ( ( *( w )->lprfsHost->forceRel )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RELAREA()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RELAREA( w, s, sp ) --> <see ( ( *( w )->lprfsHost->relArea )( w, s, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RELEVAL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RELEVAL( w, ip ) --> <see ( ( *( w )->lprfsHost->relEval )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RELTEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RELTEXT( w, s, bp ) --> <see ( ( *( w )->lprfsHost->relText )( w, s, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETREL( w, ip ) --> <see ( ( *( w )->lprfsHost->setRel )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDLSTADD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDLSTADD( w, lp ) --> <see ( ( *( w )->lprfsHost->orderListAdd )( w, lp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDLSTDELETE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDLSTDELETE( w, lp ) --> <see ( ( *( w )->lprfsHost->orderListDelete )( w, lp
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDLSTFOCUS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDLSTFOCUS( w, lp ) --> <see ( ( *( w )->lprfsHost->orderListFocus )( w, lp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDLSTREBUILD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDLSTREBUILD( w ) --> <see ( ( *( w )->lprfsHost->orderListRebuild )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDLSTCLEAR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDLSTCLEAR( w ) --> <see ( ( *( w )->lprfsHost->orderListClear )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDSETCOND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDSETCOND( w, ip ) --> <see ( ( *( w )->lprfsHost->orderCondition )( w, ip )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDCREATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDCREATE( w, ip ) --> <see ( ( *( w )->lprfsHost->orderCreate )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDDESTROY()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDDESTROY( w, p ) --> <see ( ( *( w )->lprfsHost->orderDestroy )( w, p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDINFO( w, i, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, i, p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDEXPR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDEXPR( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w,
DBOI_EXPRESSION, p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDCOND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDCOND( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_CONDITION,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDRECNO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDRECNO( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_RECNO,  p
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDPOS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDPOS( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_POSITION,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDNUMBER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDNUMBER( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_NUMBER,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDNAME( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_NAME,   p
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDBAGNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDBAGNAME( w, p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w,
DBOI_BAGNAME,p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ORDBAGEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ORDBAGEXT( w,  p ) --> <see ( ( *( w )->lprfsHost->orderInfo )( w, DBOI_BAGEXT,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLEARFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLEARFILTER( w ) --> <see ( ( *( w )->lprfsHost->clearFilter )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLEARLOCATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLEARLOCATE( w ) --> <see ( ( *( w )->lprfsHost->clearLocate )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLEARSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLEARSCOPE( w ) --> <see ( ( *( w )->lprfsHost->clearScope )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_COUNTSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_COUNTSCOPE( w, ip, lp ) --> <see ( ( *( w )->lprfsHost->countScope )( w, ip, lp
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_FILTERTEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_FILTERTEXT( w, bp ) --> <see ( ( *( w )->lprfsHost->filterText )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SCOPEINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SCOPEINFO( w, i, v ) --> <see ( ( *( w )->lprfsHost->scopeInfo )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETFILTER( w, ip ) --> <see ( ( *( w )->lprfsHost->setFilter )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETLOCATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETLOCATE( w, ip ) --> <see ( ( *( w )->lprfsHost->setLocate )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETSCOPE( w, ip ) --> <see ( ( *( w )->lprfsHost->setScope )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SKIPSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SKIPSCOPE( w, bp, l ) --> <see ( ( *( w )->lprfsHost->skipScope )( w, bp, l )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_COMPILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_COMPILE( w, bp ) --> <see ( ( *( w )->lprfsHost->compile )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_ERROR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_ERROR( w, ip ) --> <see ( ( *( w )->lprfsHost->error )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_EVALBLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_EVALBLOCK( w, v ) --> <see ( ( *( w )->lprfsHost->evalBlock )( w, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETLOCKS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETLOCKS( w, g ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_GETLOCKARRAY, g
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RAWLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RAWLOCK( w, i, l ) --> <see ( ( *( w )->lprfsHost->rawlock )( w, i, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_LOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_LOCK( w, sp ) --> <see ( ( *( w )->lprfsHost->lock )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_UNLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_UNLOCK( w, l ) --> <see ( ( *( w )->lprfsHost->unlock )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CLOSEMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CLOSEMEMFILE( w ) --> <see ( ( *( w )->lprfsHost->closeMemFile )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_CREATEMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_CREATEMEMFILE( w, bp ) --> <see ( ( *( w )->lprfsHost->createMemFile )( w, bp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETVALUEFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETVALUEFILE( w, i, bp ) --> <see ( ( *( w )->lprfsHost->getValueFile )( w, i,
bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_OPENMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_OPENMEMFILE( w, bp ) --> <see ( ( *( w )->lprfsHost->openMemFile )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_PUTVALUEFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_PUTVALUEFILE( w, i, bp ) --> <see ( ( *( w )->lprfsHost->putValueFile )( w, i,
bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_READDBHEADER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_READDBHEADER( w ) --> <see ( ( *( w )->lprfsHost->readDBHeader )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_WRITEDBHEADER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_WRITEDBHEADER( w ) --> <see ( ( *( w )->lprfsHost->writeDBHeader )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_RECSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_RECSIZE( w, lp ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_GETRECSIZE, lp
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_HEADERSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_HEADERSIZE( w, fp ) --> <see ( ( *( w )->lprfsHost->info )( w,
DBI_GETHEADERSIZE, fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_LUPDATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_LUPDATE( w, fp ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_LASTUPDATE, fp
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_SETDELIM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_SETDELIM( w, fp ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_SETDELIMITER,
fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_GETDELIM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_GETDELIM( w, fp ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_GETDELIMITER,
fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SELF_TABLEEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SELF_TABLEEXT( w, fp ) --> <see ( ( *( w )->lprfsHost->info )( w, DBI_TABLEEXT, fp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_BOF()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_BOF( w, sp ) --> <see ( ( *( SUPERTABLE )->bof )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_EOF()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_EOF( w, sp ) --> <see ( ( *( SUPERTABLE )->eof )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FOUND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FOUND( w, sp ) --> <see ( ( *( SUPERTABLE )->found )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOTO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOTO( w, l ) --> <see ( ( *( SUPERTABLE )->go )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOTOID()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOTOID( w, sp ) --> <see ( ( *( SUPERTABLE )->goToId )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOBOTTOM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOBOTTOM( w ) --> <see ( ( *( SUPERTABLE )->goBottom )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOTOP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOTOP( w ) --> <see ( ( *( SUPERTABLE )->goTop )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SEEK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SEEK( w, i1, v, i2 ) --> <see ( ( *( SUPERTABLE )->seek )( w, i1, v, i2 ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SKIP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SKIP( w, l ) --> <see ( ( *( SUPERTABLE )->skip )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SKIPFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SKIPFILTER( w, l ) --> <see ( ( *( SUPERTABLE )->skipFilter )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SKIPRAW()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SKIPRAW( w, l ) --> <see ( ( *( SUPERTABLE )->skipRaw )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ADDFIELD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ADDFIELD( w, ip ) --> <see ( ( *( SUPERTABLE )->addField )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_APPEND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_APPEND( w, l ) --> <see ( ( *( SUPERTABLE )->append )( w, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CREATEFIELDS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CREATEFIELDS( w, v ) --> <see ( ( *( SUPERTABLE )->createFields )( w, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_DELETE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_DELETE( w ) --> <see ( ( *( SUPERTABLE )->deleterec )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_DELETED()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_DELETED( w, sp ) --> <see ( ( *( SUPERTABLE )->deleted )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FIELDCOUNT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FIELDCOUNT( w, sp ) --> <see ( ( *( SUPERTABLE )->fieldCount )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FIELDDISPLAY()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FIELDDISPLAY( w, sp ) --> <see ( ( *( SUPERTABLE )->fieldDisplay )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FIELDINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FIELDINFO( w, s1, s2, v ) --> <see ( ( *( SUPERTABLE )->fieldInfo )( w, s1,
s2, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FIELDNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FIELDNAME( w, i, bp ) --> <see ( ( *( SUPERTABLE )->fieldName )( w, i, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FLUSH()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FLUSH( w ) --> <see ( ( *( SUPERTABLE )->flush )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETREC( w, bpp ) --> <see ( ( *( SUPERTABLE )->getRec )( w, bpp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETVALUE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETVALUE( w, i, v ) --> <see ( ( *( SUPERTABLE )->getValue )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETVARLEN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETVARLEN( w, i, lp ) --> <see ( ( *( SUPERTABLE )->getVarLen )( w, i, lp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOCOLD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOCOLD( w ) --> <see ( ( *( SUPERTABLE )->goCold )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GOHOT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GOHOT( w ) --> <see ( ( *( SUPERTABLE )->goHot )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_PUTVALUE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_PUTVALUE( w, i, v ) --> <see ( ( *( SUPERTABLE )->putValue )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_PUTREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_PUTREC( w, bp ) --> <see ( ( *( SUPERTABLE )->putRec )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RECALL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RECALL( w ) --> <see ( ( *( SUPERTABLE )->recall )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RECCOUNT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RECCOUNT( w, sp ) --> <see ( ( *( SUPERTABLE )->reccount )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RECINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RECINFO( w, v1, i, v2 ) --> <see ( ( *( SUPERTABLE )->recInfo )( w, v1, i, v2
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RECNO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RECNO( w, sp ) --> <see ( ( *( SUPERTABLE )->recno )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETFIELDEXTENT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETFIELDEXTENT( w, s ) --> <see ( ( *( SUPERTABLE )->setFieldExtent )( w, s )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ALIAS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ALIAS( w, bp ) --> <see ( ( *( SUPERTABLE )->alias )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLOSE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLOSE( w ) --> <see ( ( *( SUPERTABLE )->close )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CREATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CREATE( w, ip ) --> <see ( ( *( SUPERTABLE )->create )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_INFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_INFO( w, i, g ) --> <see ( ( *( SUPERTABLE )->info )( w, i, g ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_NEW()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_NEW( w ) --> <see ( ( *( SUPERTABLE )->newarea )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_OPEN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_OPEN( w, ip ) --> <see ( ( *( SUPERTABLE )->open )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RELEASE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RELEASE( w ) --> <see ( ( *( SUPERTABLE )->release )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_STRUCTSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_STRUCTSIZE( w, sp ) --> <see ( ( *( SUPERTABLE )->structSize )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SYSNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SYSNAME( w, bp ) --> <see ( ( *( SUPERTABLE )->sysName )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_DBEVAL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_DBEVAL( w, ip ) --> <see ( ( *( SUPERTABLE )->dbEval )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_PACK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_PACK( w ) --> <see ( ( *( SUPERTABLE )->pack )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_PACKREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_PACKREC( w, l, sp ) --> <see ( ( *( SUPERTABLE )->packRec )( w, l, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SORT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SORT( w, ip ) --> <see ( ( *( SUPERTABLE )->sort )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_TRANS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_TRANS( w, ip ) --> <see ( ( *( SUPERTABLE )->trans )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_TRANSREC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_TRANSREC( w, ip ) --> <see ( ( *( SUPERTABLE )->transRec )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ZAP()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ZAP( w ) --> <see ( ( *( SUPERTABLE )->zap )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CHILDEND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CHILDEND( w, ip ) --> <see ( ( *( SUPERTABLE )->childEnd )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CHILDSTART()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CHILDSTART( w, ip ) --> <see ( ( *( SUPERTABLE )->childStart )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CHILDSYNC()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CHILDSYNC( w, ip ) --> <see ( ( *( SUPERTABLE )->childSync )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SYNCCHILDREN()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SYNCCHILDREN( w ) --> <see ( ( *( SUPERTABLE )->syncChildren )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLEARREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLEARREL( w ) --> <see ( ( *( SUPERTABLE )->clearRel )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FORCEREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FORCEREL( w ) --> <see ( ( *( SUPERTABLE )->forceRel )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RELAREA()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RELAREA( w, s, sp ) --> <see ( ( *( SUPERTABLE )->relArea )( w, s, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RELEVAL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RELEVAL( w, ip ) --> <see ( ( *( SUPERTABLE )->relEval )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RELTEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RELTEXT( w, s, bp ) --> <see ( ( *( SUPERTABLE )->relText )( w, s, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETREL()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETREL( w, ip ) --> <see ( ( *( SUPERTABLE )->setRel )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDLSTADD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDLSTADD( w, lp ) --> <see ( ( *( SUPERTABLE )->orderListAdd )( w, lp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDLSTDELETE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDLSTDELETE( w, lp ) --> <see ( ( *( SUPERTABLE )->orderListDelete )( w, lp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDLSTFOCUS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDLSTFOCUS( w, lp ) --> <see ( ( *( SUPERTABLE )->orderListFocus )( w, lp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDLSTREBUILD()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDLSTREBUILD( w ) --> <see ( ( *( SUPERTABLE )->orderListRebuild )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDLSTCLEAR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDLSTCLEAR( w ) --> <see ( ( *( SUPERTABLE )->orderListClear )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDSETCOND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDSETCOND( w, ip ) --> <see ( ( *( SUPERTABLE )->orderCondition )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDCREATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDCREATE( w, ip ) --> <see ( ( *( SUPERTABLE )->orderCreate )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDDELETE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDDELETE( w, ip ) --> <see ( ( *( SUPERTABLE )->orderDelete )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDINFO( w, i, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, i, p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDEXPR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDEXPR( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_EXPRESSION,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDCOND()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDCOND( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_CONDITION,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDRECNO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDRECNO( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_RECNO,  p
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDPOS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDPOS( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_POSITION,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDNUMBER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDNUMBER( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_NUMBER, p
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDNAME( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_NAME,   p )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDBAGNAME()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDBAGNAME( w, p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w,
DBOI_BAGNAME,p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ORDBAGEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ORDBAGEXT( w,  p ) --> <see ( ( *( SUPERTABLE )->orderInfo )( w, DBOI_BAGEXT,
p ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLEARFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLEARFILTER( w ) --> <see ( ( *( SUPERTABLE )->clearFilter )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLEARLOCATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLEARLOCATE( w ) --> <see ( ( *( SUPERTABLE )->clearLocate )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLEARSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLEARSCOPE( w ) --> <see ( ( *( SUPERTABLE )->clearScope )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_COUNTSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_COUNTSCOPE( w, ip, lp ) --> <see ( ( *( SUPERTABLE )->countScope )( w, ip, lp
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_FILTERTEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_FILTERTEXT( w, bp ) --> <see ( ( *( SUPERTABLE )->filterText )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SCOPEINFO()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SCOPEINFO( w, i, v ) --> <see ( ( *( SUPERTABLE )->scopeInfo )( w, i, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETFILTER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETFILTER( w, ip ) --> <see ( ( *( SUPERTABLE )->setFilter )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETLOCATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETLOCATE( w, ip ) --> <see ( ( *( SUPERTABLE )->setLocate )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETSCOPE( w, ip ) --> <see ( ( *( SUPERTABLE )->setScope )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SKIPSCOPE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SKIPSCOPE( w, bp, l ) --> <see ( ( *( SUPERTABLE )->skipScope )( w, bp, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_COMPILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_COMPILE( w, bp ) --> <see ( ( *( SUPERTABLE )->compile )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_ERROR()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_ERROR( w, ip ) --> <see ( ( *( SUPERTABLE )->error )( w, ip ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_EVALBLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_EVALBLOCK( w, v ) --> <see ( ( *( SUPERTABLE )->evalBlock )( w, v ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETLOCKS()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETLOCKS( w, g ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_GETLOCKARRAY, g
) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RAWLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RAWLOCK( w, i, l ) --> <see ( ( *( SUPERTABLE )->rawlock )( w, i, l ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_LOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_LOCK( w, sp ) --> <see ( ( *( SUPERTABLE )->lock )( w, sp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_UNLOCK()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_UNLOCK( w ) --> <see ( ( *( SUPERTABLE )->unlock )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CLOSEMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CLOSEMEMFILE( w ) --> <see ( ( *( SUPERTABLE )->closeMemFile )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_CREATEMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_CREATEMEMFILE( w, bp ) --> <see ( ( *( SUPERTABLE )->createMemFile )( w, bp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETVALUEFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETVALUEFILE( w, i, bp ) --> <see ( ( *( SUPERTABLE )->getValueFile )( w, i,
bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_OPENMEMFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_OPENMEMFILE( w, bp ) --> <see ( ( *( SUPERTABLE )->openMemFile )( w, bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_PUTVALUEFILE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_PUTVALUEFILE( w, i, bp ) --> <see ( ( *( SUPERTABLE )->putValueFile )( w, i,
bp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_READDBHEADER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_READDBHEADER( w ) --> <see ( ( *( SUPERTABLE )->readDBHeader )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_WRITEDBHEADER()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_WRITEDBHEADER( w ) --> <see ( ( *( SUPERTABLE )->writeDBHeader )( w ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_RECSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_RECSIZE( w, lp ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_GETRECSIZE, lp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_HEADERSIZE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_HEADERSIZE( w, fp ) --> <see ( ( *( SUPERTABLE )->info )( w,
DBI_GETHEADERSIZE, fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_LUPDATE()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_LUPDATE( w, fp ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_LASTUPDATE, fp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_SETDELIM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_SETDELIM( w, fp ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_SETDELIMITER,
fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_GETDELIM()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_GETDELIM( w, fp ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_GETDELIMITER,
fp ) )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

SUPER_TABLEEXT()

Syntax

C Prototype (macro definition)

```
#include <hbapirdd.h>
SUPER_TABLEEXT( w, fp ) --> <see ( ( *( SUPERTABLE )->info )( w, DBI_TABLEEXT, fp )
)>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is hbapirdd.h

Platforms

All

_evalLaunch()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_evalLaunch --> <see hb_evalLaunch>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_evalLaunch()

_evalNew()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_evalNew --> <see hb_evalNew>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_evalNew()

_evalPutParam()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_evalPutParam --> <see hb_evalPutParam>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_evalPutParam()

_evalRelease()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_evalRelease --> <see hb_evalRelease>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_evalRelease()

_itemArrayGet()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemArrayGet --> <see hb_itemArrayGet>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemArrayGet()

_itemArrayNew()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemArrayNew --> <see hb_itemArrayNew>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemArrayNew()

_itemArrayPut()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemArrayPut --> <see hb_itemArrayPut>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemArrayPut()

_itemNew()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemNew --> <see hb_itemNew>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemNew()

_itemParam()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemParam --> <see hb_itemParam>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemParam()

_itemRelease()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemRelease --> <see hb_itemRelease>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemRelease()

_itemReturn()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemReturn --> <see hb_itemReturn>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemReturn()

_itemSize()

Syntax

C Prototype (macro replacement)

```
#include <item.api>
_itemSize --> <see hb_itemSize>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemSize()

_itemType()

Syntax

```
C Prototype (macro replacement)

#include <item.api>
_itemType --> <see hb_itemType>
```

Arguments

Returns

Description

Status

Ready

Header file is item.api

Platforms

All

See Also:

hb_itemType()

_reta()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_reta --> <see hb_reta>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_reta()

_pcount()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_pcount --> <see hb_pcount>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_pcount()

_tchdir()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tchdir --> <see hb_fsChDir>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsChDir()

_tchdrv()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tchdrv --> <see hb_fsChDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsChDrv()

_tclose()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tclose --> <see hb_fsClose>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsClose()

_tcommit()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tcommit --> <see hb_fsCommit>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsCommit()

_tcreat()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tcreat --> <see hb_fsCreate>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsCreate()

_tcurdir()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tcurdir --> <see hb_fsCurDir>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsCurDir()

_tcurdrv()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tcurdrv --> <see hb_fsCurDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsCurDrv()

_tdevraw()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tdevraw --> <see hb_fsSetDevRaw>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsSetDevRaw()

_terror()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_terror --> <see hb_fsError>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsError()

_tisdevice()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tisdevice --> <see hb_fsIsDevice>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsIsDevice()

_tisdrv()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tisdrv --> <see hb_fsIsDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsIsDrv()

_tlock()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tlock --> <see hb_fsLock>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsLock()

_tlseek()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tlseek --> <see hb_fsSeek>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsSeek()

_tmkdir()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tmkdir --> <see hb_fsMkDir>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsMkDir()

_topen()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_topen --> <see hb_fsOpen>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsOpen()

_tread()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tread --> <see hb_fsRead>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsRead()

_trename()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_trename --> <see hb_fsRename>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsRename()

_trmdir()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_trmdir --> <see hb_fsRmDir>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsRmDir()

_tunlink()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_tunlink --> <see hb_fsDelete>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsDelete()

_twrite()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_twrite --> <see hb_fsWrite>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

hb_fsWrite()

_bset()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_bset --> <see memset>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

ARRAY()

_bmove()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_bmove --> <see memmove>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

ARRAY()

_bcopy()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_bcopy --> <see memcpy>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

ARRAY()

_bcmp()

Syntax

C Prototype (macro replacement)

```
#include <hbundoc.api>
_bcmp --> <see memcmp>
```

Arguments

Returns

Description

Status

Ready

Header file is hbundoc.api

Platforms

All

See Also:

ARRAY()

_gtBox()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtBox --> <see hb_gtBox>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtBox()

_gtColorSelect()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtColorSelect --> <see hb_gtColorSelect>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtColorSelect()

_gtDispBegin()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtDispBegin --> <see hb_gtDispBegin>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtDispBegin()

_gtDispCount()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtDispCount --> <see hb_gtDispCount>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtDispCount()

_gtDispEnd()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtDispEnd --> <see hb_gtDispEnd>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtDispEnd()

_gtGetColorStr()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtGetColorStr --> <see hb_gtGetColorStr>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtGetColorStr()

_gtGetCursor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtGetCursor --> <see hb_gtGetCursor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtGetCursor()

_gtGetPos()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtGetPos --> <see hb_gtGetPos>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtGetPos()

_gtIsColor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtIsColor --> <see hb_gtIsColor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtIsColor()

_gtMaxCol()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtMaxCol --> <see hb_gtMaxCol>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtMaxCol()

_gtMaxRow()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtMaxRow --> <see hb_gtMaxRow>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtMaxRow()

_gtPostExt()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtPostExt --> <see hb_gtPostExt>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtPostExt()

_gtPreExt()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtPreExt --> <see hb_gtPreExt>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtPreExt()

_gtRectSize()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtRectSize --> <see hb_gtRectSize>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtRectSize()

_gtRepChar()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtRepChar --> <see hb_gtRepChar>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtRepChar()

_gtRest()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtRest --> <see hb_gtRest>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtRest()

_gtSave()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSave --> <see hb_gtSave>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSave()

_gtScrDim()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtScrDim --> <see hb_gtScrDim>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtScrDim()

_gtScroll()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtScroll --> <see hb_gtScroll>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtScroll()

_gtSetBlink()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetBlink --> <see hb_gtSetBlink>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetBlink()

_gtSetColorStr()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetColorStr --> <see hb_gtSetColorStr>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetColorStr()

_gtSetCursor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetCursor --> <see hb_gtSetCursor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetCursor()

_gtSetMode()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetMode --> <see hb_gtSetMode>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetMode()

_gtSetPos()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetPos --> <see hb_gtSetPos>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetPos()

_gtSetSnowFlag()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetSnowFlag --> <see hb_gtSetSnowFlag>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetSnowFlag()

_gtWrite()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWrite --> <see hb_gtWrite>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWrite()

_gtWriteAt()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWriteAt --> <see hb_gtWriteAt>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWriteAt()

_gtWriteCon()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWriteCon --> <see hb_gtWriteCon>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWriteCon()

_gtInit()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtInit --> <see hb_gtInit>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtInit()

_gtExit()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtExit --> <see hb_gtExit>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtExit()

_gtWCreate()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWCreate --> <see hb_gtWCreate>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWCreate()

_gtWDestroy()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWDestroy --> <see hb_gtWDestroy>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWDestroy()

_gtWFlash()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWFlash --> <see hb_gtWFlash>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWFlash()

_gtWApp()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWApp --> <see hb_gtWApp>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWApp()

_gtWCurrent()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWCurrent --> <see hb_gtWCurrent>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWCurrent()

_gtWPos()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWPos --> <see hb_gtWPos>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWPos()

_gtWVis()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtWVis --> <see hb_gtWVis>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtWVis()

_gtModalRead()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtModalRead --> <see hb_gtModalRead>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtModalRead()

_gtBeginWrite()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtBeginWrite --> <see hb_gtBeginWrite>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtBeginWrite()

_gtEndWrite()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtEndWrite --> <see hb_gtEndWrite>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtEndWrite()

_gtFlushCursor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtFlushCursor --> <see hb_gtFlushCursor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtFlushCursor()

_gtSetColor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetColor --> <see hb_gtSetColor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetColor()

_gtGetColor()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtGetColor --> <see hb_gtGetColor>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtGetColor()

_gtSetBorder()

Syntax

C Prototype (macro replacement)

```
#include <gt.api>
_gtSetBorder --> <see hb_gtSetBorder>
```

Arguments

Returns

Description

Status

Ready

Header file is gt.api

Platforms

All

See Also:

hb_gtSetBorder()

_xalloc()

Syntax

C Prototype (macro replacement)

```
#include <fm.api>
_xalloc --> <see hb_xalloc>
```

Arguments

Returns

Description

Status

Ready

Header file is fm.api

Platforms

All

See Also:

hb_xalloc()

_xgrab()

Syntax

C Prototype (macro replacement)

```
#include <fm.api>
_xgrab --> <see hb_xgrab>
```

Arguments

Returns

Description

Status

Ready

Header file is fm.api

Platforms

All

See Also:

hb_xgrab()

_xfree()

Syntax

C Prototype (macro replacement)

```
#include <fm.api>
_xfree --> <see hb_xfree>
```

Arguments

Returns

Description

Status

Ready

Header file is fm.api

Platforms

All

See Also:

hb_xfree()

_exmgrab()

Syntax

C Prototype (macro replacement)

```
#include <fm.api>
_exmgrab --> <see hb_xgrab>
```

Arguments

Returns

Description

Status

Ready

Header file is fm.api

Platforms

All

See Also:

hb_xgrab()

_fsChDir()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsChDir --> <see hb_fsChDir>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsChDir()

_fsChDrv()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsChDrv --> <see hb_fsChDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsChDrv()

_fsClose()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsClose --> <see hb_fsClose>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsClose()

_fsCommit()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsCommit --> <see hb_fsCommit>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsCommit()

_fsCreate()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsCreate --> <see hb_fsCreate>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsCreate()

_fsCurDir()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsCurDir --> <see hb_fsCurDir>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsCurDir()

_fsCurDrv()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsCurDrv --> <see hb_fsCurDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsCurDrv()

_fsDelete()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsDelete --> <see hb_fsDelete>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsDelete()

_fsError()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsError --> <see hb_fsError>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsError()

_fsExtOpen()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsExtOpen --> <see hb_fsExtOpen>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsExtOpen()

_fsIsDrv()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsIsDrv --> <see hb_fsIsDrv>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsIsDrv()

_fsLock()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsLock --> <see hb_fsLock>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsLock()

_fsMkDir()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsMkDir --> <see hb_fsMkDir>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsMkDir()

_fsOpen()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsOpen --> <see hb_fsOpen>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsOpen()

_fsRead()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsRead --> <see hb_fsRead>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsRead()

_fsRmDir()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsRmDir --> <see hb_fsRmDir>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsRmDir()

_fsRename()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsRename --> <see hb_fsRename>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsRename()

_fsSeek()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsSeek --> <see hb_fsSeek>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsSeek()

_fsWrite()

Syntax

C Prototype (macro replacement)

```
#include <filesys.api>
_fsWrite --> <see hb_fsWrite>
```

Arguments

Returns

Description

Status

Ready

Header file is filesys.api

Platforms

All

See Also:

hb_fsWrite()

ALENGTH

Syntax

C Prototype (macro definition)

```
#include <extend.api>
ALENGTH( n ) --> <see hb_parinfa( n, 0 )>
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Header file is extend.api

Platforms

All

_parc()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parc --> <see hb_parc>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parc()

_parclen()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parclen --> <see hb_parclen>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parclen()

_parcsiz()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parcsiz --> <see hb_parcsiz>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parcsiz()

_pards()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_pards --> <see hb_pards>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_pards()

_parinfa()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parinfa --> <see hb_parinfa>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parinfa()

_parinfo()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parinfo --> <see hb_parinfo>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parinfo()

_parl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parl --> <see hb_parl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parl()

_parnd()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parnd --> <see hb_parnd>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parnd()

_parni()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parni --> <see hb_parni>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parni()

_parnl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_parnl --> <see hb_parnl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_parnl()

_ret()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_ret --> <see hb_ret>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_ret()

_retc()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retc --> <see hb_retc>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retc()

_retclen()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retclen --> <see hb_retclen>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retclen()

_retds()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retds --> <see hb_retds>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retds()

_retl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retl --> <see hb_retl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retl()

_retnd()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retnd --> <see hb_retnd>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retnd()

_retni()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retni --> <see hb_retni>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retni()

_retnl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_retnl --> <see hb_retnl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_retnl()

_storc()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_storc --> <see hb_storc>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_storc()

_storclen()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_storclen --> <see hb_storclen>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_storclen()

_stords()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_stords --> <see hb_stords>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_stords()

_storl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_storl --> <see hb_storl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_storl()

_stornd()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_stornd --> <see hb_stornd>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_stornd()

_storni()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_storni --> <see hb_storni>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_storni()

_stornl()

Syntax

C Prototype (macro replacement)

```
#include <extend.api>
_stornl --> <see hb_stornl>
```

Arguments

Returns

Description

Status

Ready

Header file is extend.api

Platforms

All

See Also:

hb_stornl()

_errGetDescription()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetDescription --> <see hb_errGetDescription>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetDescription()

_errGetFileName()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetFileName --> <see hb_errGetFileName>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetFileName()

_errGetFlags()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetFlags --> <see hb_errGetFlags>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetFlags()

_errGetGenCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetGenCode --> <see hb_errGetGenCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetGenCode()

_errGetOperation()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetOperation --> <see hb_errGetOperation>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetOperation()

_errGetOsCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetOsCode --> <see hb_errGetOsCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetOsCode()

_errGetSeverity()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetSeverity --> <see hb_errGetSeverity>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetSeverity()

_errGetSubCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetSubCode --> <see hb_errGetSubCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetSubCode()

_errGetSubSystem()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetSubSystem --> <see hb_errGetSubSystem>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetSubSystem()

_errGetTries()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errGetTries --> <see hb_errGetTries>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errGetTries()

_errLaunch()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errLaunch --> <see hb_errLaunch>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errLaunch()

_errNew()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errNew --> <see hb_errNew>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errNew()

_errPutDescription()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutDescription --> <see hb_errPutDescription>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutDescription()

_errPutFileName()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutFileName --> <see hb_errPutFileName>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutFileName()

_errPutFlags()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutFlags --> <see hb_errPutFlags>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutFlags()

_errPutGenCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutGenCode --> <see hb_errPutGenCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutGenCode()

_errPutOperation()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutOperation --> <see hb_errPutOperation>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutOperation()

_errPutOsCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutOsCode --> <see hb_errPutOsCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutOsCode()

_errPutSeverity()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutSeverity --> <see hb_errPutSeverity>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutSeverity()

_errPutSubCode()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutSubCode --> <see hb_errPutSubCode>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutSubCode()

_errPutSubSystem()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutSubSystem --> <see hb_errPutSubSystem>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutSubSystem()

_errPutTries()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errPutTries --> <see hb_errPutTries>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errPutTries()

_errRelease()

Syntax

C Prototype (macro replacement)

```
#include <error.api>
_errRelease --> <see hb_errRelease>
```

Arguments

Returns

Description

Status

Ready

Header file is error.api

Platforms

All

See Also:

hb_errRelease()

hb_dateSeconds()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateSeconds( void ) --> ( double )dResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateToday()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateToday( long * plYear, long * plMonth, long * plDay ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateTimeStr()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateTimeStr( char * pszTime ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateCMonth()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateCMonth( int iMonth ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateCDOW()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateCDOW( int iDay ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateDOW()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateDOW( long lYear, long lMonth, long lDay ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateFormat()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateFormat( const char * szDate, char * szFormattedDate, const char *
szDateFormat ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateEncode()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateEncode( long lYear, long lMonth, long lDay ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateDecode()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateDecode( long julian, long * plYear, long * plMonth, long * plDay ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateStrPut()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateStrPut( char * szDate, long lYear, long lMonth, long lDay ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateStrGet()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateStrGet( const char * szDate, long * plYear, long * plMonth, long * plDay )
--> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateDecStr()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateDecStr( char * szDate, long lJulian ) --> ( char * )pszResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_dateEncStr()

Syntax

C Prototype

```
#include <hbdate.h>
hb_dateEncStr( char * szDate ) --> ( long )lResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_macroError()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_macroError( int iError, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_macroYYParse()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_macroYYParse( HB_MACRO_PTR pMacro ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPCode1()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPCode1( BYTE byte, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPCode2()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPCode2( BYTE byte1, BYTE byte2, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPCode3()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPCode3( BYTE byte1, BYTE byte2, BYTE byte3, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPCode4()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPCode4( BYTE byte1, BYTE byte2, BYTE byte3, BYTE byte4, HB_BISON_PTR
pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPCodeN()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPCodeN( BYTE * pBuffer, ULONG ulSize, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compLocalVarGetPos()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compLocalVarGetPos( char * szVarName, HB_BISON_PTR pMacro ) --> ( int )iResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenJump()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenJump( LONG lOffset, HB_BISON_PTR pMacro ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenJumpFalse()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenJumpFalse( LONG lOffset, HB_BISON_PTR pMacro ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenJumpThere()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenJumpThere( ULONG ulFrom, ULONG ulTo, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenJumpHere()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenJumpHere( ULONG ulOffset, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenJumpTrue()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenJumpTrue( LONG lOffset, HB_BISON_PTR pMacro ) --> ( ULONG )ulResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compMemvarGenPCode()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compMemvarGenPCode( BYTE bPCode, char * szVarName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushSymbol()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushSymbol( char * szSymbolName, int isFunction, HB_BISON_PTR pMacro ) -->
void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushLong()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushLong( long lNumber, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenMessage()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenMessage( char * szMsgName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenMessageData()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenMessageData( char * szMsg, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPopVar()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPopVar( char * szVarName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPopAliasedVar()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPopAliasedVar( char * szVarName, BOOL bPushAliasValue, char * szAlias,
long lWorkarea, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushVar()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushVar( char * szVarName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushVarRef()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushVarRef( char * szVarName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushAliasedVar()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushAliasedVar( char * szVarName, BOOL bPushAliasValue, char * szAlias,
long lWorkarea, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushLogical()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushLogical( int iTrueFalse, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushDouble()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushDouble( double dNumber, BYTE bWidth, BYTE bDec, HB_BISON_PTR pMacro )
--> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushFunCall()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushFunCall( char * szFunName, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compGenPushString()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compGenPushString( char * szText, ULONG ulStrLen, HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compCodeBlockStart()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compCodeBlockStart( HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_compCodeBlockEnd()

Syntax

C Prototype

```
#include <hbmacro.h>
hb_compCodeBlockEnd( HB_BISON_PTR pMacro ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is macro

Platforms

All

hb_setInitialize()

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_setRelease()

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_setListenerAdd()

Syntax

C Prototype

```
#include <hbset.h>
hb_setListenerAdd( PHB_SET_LISTENER_CALLBACK callback ) --> int
```

Arguments

value. The first parameter identifies the SET parameter that is to be changed and the second parameter identifies whether the call is from before or after the value is changed. The callback function will be called twice whenever a SET parameter is changed using the Harbour SET function. The first call takes place before the SET value is changed and the second one is after the SET parameter has been changed.

Returns

deactivate the callback function.

Description

This function allows a subsystem that needs to track the status of some SET parameters to be notified whenever a SET parameter gets changed.

Examples

```
void callback_function( HB_set_enum set, HB_set_listener_enum when )
{
   printf("\nCalled for SET parameter %d %s changing.",
      set, (when ? "after" : "before"));
}
int handle = hb_setListenerAdd( callback_function );
```

Status

Ready

Compliance

Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

See Also:

hb_setListenerRemove()

hb_setListenerNotify()

Syntax

C Prototype

```
#include <hbset.h>
hb_setListenerNotify( HB_set_enum set, HB_set_listener_enum
when ) --> int
```

Arguments

HB_SET_LISTENER_BEFORE when called before the SET parameter is to be changed and set to HB_SET_LISTENER_AFTER when called after the SET parameter has been changed.

Returns

Description

This function notifies all SET listener callback functions. It must be called any time you change the value of a SET parameter directly instead of using the Harbour SET function. Both before and after the change.

Examples

```
hb_setListenerNotify( HB_SET_DECIMALS, HB_SET_LISTENER_BEFORE );
hb_set.HB_SET_DECIMALS = 3;
hb_setListenerNotify( HB_SET_DECIMALS, HB_SET_LISTENER_AFTER );
```

Status

Ready

Compliance

Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

See Also:

hb_setListenerAdd()

hb_setListenerRemove()

Syntax

C Prototype

```
#include <hbset.h>
hb_setListenerRemove( int handle ) --> int
```

Arguments

Returns

the handle if the callback function was removed.

Description

This function removes a SET listener callback function.

Examples

```
int handle = hb_setListenerAdd( callback_function );
...
hb_setListenerRemove( handle );
```

Status

Ready

Compliance

Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

See Also:

hb_setListenerAdd()

hb_vmInit()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmInit( BOOL bStartMainProc ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmQuit()

Immediately quits the virtual machine

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmQuit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmExecute()

Invokes the virtual machine

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmExecute( const BYTE * pCode, PHB_SYMB pSymbols ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmProcessSymbols()

Statics symbols initialization

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmProcessSymbols( PHB_SYMB pSymbols, USHORT uiSymbols ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmSymbolInit_RT()

Initialization of runtime support symbols

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmSymbolInit_RT( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmRequestQuit()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmRequestQuit( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmRequestEndProc()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmRequestEndProc( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmRequestCancel()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmRequestCancel( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmRequestBreak()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmRequestBreak( PHB_ITEM pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmRequestQuery()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmRequestQuery( void ) --> ( USHORT )usResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmMessage()

Sends a message to an object

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmMessage( PHB_SYMB pSymMsg ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmDo()

Invoke the virtual machine

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmDo( USHORT uiParams ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmFunction()

Executes a function saving its result

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmFunction( USHORT uiParams ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmSend()

Sends a message to an object

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmSend( USHORT uiParams ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmEvalBlock()

Executes passed codeblock with no arguments

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmEvalBlock( PHB_ITEM pBlockItem ) --> ( PHB_ITEM )pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmEvalBlockV()

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmEvalBlockV( PHB_ITEM pBlockItem, USHORT uiArgCount, ... ) --> ( PHB_ITEM
)pResult
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPush()

Pushes a generic item onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPush( PHB_ITEM pItem ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushNil()

In this case it places nil at self

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushNil( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushNumber()

Pushes a number on to the stack and decides if it is integer, long or double

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushNumber( double dNumber, int iDec ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushInteger()

Pushes a integer number onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushInteger( int iNumber ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushLong()

Pushes a long number onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushLong( long lNumber ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushDouble()

Pushes a double number onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushDouble( double lNumber, int iDec ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushLogical()

Pushes a logical value onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushLogical( BOOL bValue ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushString()

Pushes a string on to the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushString( char * szText, ULONG length ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushDate()

Pushes a long date onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushDate( long lDate ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushSymbol()

Pushes a function pointer onto the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushSymbol( PHB_SYMB pSym ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_vmPushPointer()

Push an item of HB_IT_POINTER type

Syntax

C Prototype

```
#include <hbvm.h>
hb_vmPushPointer( void * ) --> void
```

Arguments

<void *>

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stackDispCall()

Syntax

C Prototype

```
#include <hbvm.h>
hb_stackDispCall( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

hb_stackPop()

Pops an item from the stack

Syntax

C Prototype

```
#include <hbvm.h>
hb_stackPop( void ) --> void
```

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is vm

Platforms

All

PROCNAME()

Gets the name of the current function on the stack

Syntax

PROCNAME(<nLevel>) --> <cProcName>

Arguments

<nLevel> is the function level required.

Returns

<cProcName> The name of the function that it is being executed.

Description

This function looks at the top of the stack and gets the current executed function if no arguments are passed. Otherwise it returns the name of the function or procedure at <nLevel>.

Examples

See Test

Tests

```
This test will show the functions and procedures in stack.
before executing it.
function Test()
   LOCAL n := 1
   while !Empty( ProcName( n ) )
      ? ProcName( n++ )
   end do
return nil
```

Status

Ready

Compliance

PROCNAME() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

PROCLINE()
PROCFILE()

PROCLINE()

Gets the line number of the current function on the stack.

Syntax

PROCLINE(<nLevel>) --> <nLine>

Arguments

<nLevel> is the function level required.

Returns

<nLine> The line number of the function that it is being executed.

Description

This function looks at the top of the stack and gets the current line number of the executed function if no arguments are passed. Otherwise it returns the line number of the function or procedure at <nLevel>.

Examples

See Test

Tests

```
function Test()
   ? ProcLine( 0 )
   ? ProcName( 2 )
return nil
```

Status

Ready

Compliance

PROCLINE() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

PROCNAME()
PROCFILE()

PROCFILE()

This function allways returns an empty string.

Syntax

```
PROCFILE( <xExp> ) --> <cEmptyString>
```

Arguments

<xExp> is any valid type.

Returns

<cEmptyString> Return an empty string

Description

This function is added to the RTL for full compatibility. It always returns an empty string.

Examples

```
? ProcFile()
```

Tests

```
function Test()
   ? ProcFile()
   ? ProcFile( NIL )
   ? ProcFile( 2 )
return nil
```

Status

Ready

Compliance

PROCFILE() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

PROCNAME()
PROCLINE()

HB_PVALUE()

Retrieves the value of an argument.

Syntax

```
HB_PVALUE( <nArg> ) --> <xExp>
```

Arguments

Returns

<xExp> Returns the value stored by an argument.

Description

This function is useful to check the value stored in an argument.

Examples

```
See Test
```

Tests

```
function Test( nValue, cString )
   if PCount() == 2
      ? hb_PValue( 1 ), nValue
      ? hb_PValue( 2 ), cString
   endif
return nil
```

Status

Ready

Compliance

HB_PVALUE() is a new function and hence not CA-Clipper compliant.

Files

Library is vm

See Also:

PCOUNT()

PCOUNT()

Retrieves the number of arguments passed to a function.

Syntax

PCOUNT() --> <nArgs>

Arguments

Returns

<nArgs> A number that indicates the number of arguments passed to a function or procedure.

Description

This function is useful to check if a function or procedure has received the required number of arguments.

Examples

See Test

Tests

```
function Test( xExp )
   if PCount() == 0
      ? "This function needs a parameter"
   else
      ? xExp
   endif
return nil
```

Status

Ready

Compliance

PCOUNT() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

HB_PVALUE()

__QUIT()

Terminates an application.

Syntax

`__QUIT() --> NIL`

Arguments

Returns

Description

This function terminates the current application and returns to the system.

Examples

See Test

Tests

```
function EndApp( lYesNo )
   if lYesNo
      __Quit()
   endif
return nil
```

Status

Ready

Compliance

__QUIT() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

ARRAY()

CLIPINIT()

Initialize various Harbour sub-systems

Syntax

```
CLIPINIT() --> NIL
```

Arguments

Returns

CLIPINIT() always return NIL.

Description

CLIPINIT() is one of the pre-defined INIT PROCEDURE and is executed at program startup. It declare an empty MEMVAR PUBLIC array called GetList that is going to be used by the Get system. It activates the default error handler, and (at least for the moment) calls the function that sets the default help key.

Status

Ready

Compliance

It is said that CLIPINIT() should not call the function that sets the default help key since CA-Clipper does it in some other place.

Platforms

All

See Also:

ARRAY()

__SetHelpK()

Set F1 as the default help key

Syntax

```
__SetHelpK() --> NIL
```

Arguments

Returns

__SetHelpK() always return NIL.

Description

Set F1 to execute a function called HELP if such a function is linked into the program.

Status

Ready

Compliance

__SetHelpK() works exactly like CA-Clipper's __SetHelpK()

Files

Library is vm

See Also:

__XHELP()
SET KEY
SETKEY()

BREAK()

Exits from a BEGIN SEQUENCE block

Syntax

BREAK(<xExp>) --> NIL

Arguments

<xExp> is any valid expression. It is always required. If do not want to pass any argument, just use NIL.

Returns

Description

This function passes control to the RECOVER statement in a BEGIN SEQUENCE block.

Examples

```
Break( NIL )
```

Status

Ready

Compliance

BREAK() is fully CA-Clipper compliant.

Files

Library is vm

See Also:

ARRAY()

DO()

Calls a procedure or a function

Syntax

DO(<xFuncProc> [, <xArguments...>])

Arguments

<xFuncProc> = Either a string with a function/procedure name to be called or a codeblock to evaluate.
<xArguments> = arguments passed to a called function/procedure or to a codeblock.

Returns

Description

This function can be called either by the harbour compiler or by user. The compiler always passes the item of IT_SYMBOL type that stores the name of procedure specified in DO <proc> WITH ... statement.
If called procedure/function doesn't exist then a runtime error is generated.
This function can be used as replacement of macro operator. It is also used internally to implement DO <proc> WITH <args...> In this case <xFuncProc> is of type HB_SYMB.

Examples

```
cbCode ={|x| MyFunc( x )}
DO( cbCode, 1 )

cFunction := "MyFunc"
xRetVal :=DO( cFunction, 2 )

Old style (slower):
DO &cFunction WITH 3
```

Files

Library is rtl

__VMVARLGET()

Retrive a local variable from a procedure level

Syntax

__VMVARLGET(<nProcLevel>, <nLocal>)

Arguments

<nProcLevel> Is the procedure level, same as used in ProcName() and ProcLine(), from which a local variable containts is going to be retrieved.
<nLocal> Is the index of the local variable to retrieve.

Returns

Description

This function is used from the debugger

Files

Library is vm

The idle states

Read me file for Idle States

Description

The idle state is the state of the harbour virtual machine when it waits for the user input from the keyboard or the mouse. The idle state is entered during INKEY() calls currently. All applications that don't use INKEY() function call can signal the idle states with the call of HB_IDLESTATE() function (or hb_idleState() on C level).

During idle states the virtual machine calls the garbage collector and it can call user defined actions (background tasks). It also releases the CPU time slices for some poor platforms that are not smart enough (Windows NT).

For defining the background tasks see the HB_IDLEADD() and HB_IDLEDEL() functions.

For direct call for background actions see HB_IDLESTATE() function.

For signaling the idle state from C code see the hb_idleState() API function.

See Also:

HB_IDLEADD()
HB_IDLEDEL()

HB_IDLEADD()

Adds the background task.

Syntax

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

Arguments

<cbAction> is a codeblock that will be executed during idle states. There are no arguments passed to this codeblock during evaluation.

Returns

<nHandle> The handle (an integer value) that identifies the task. This handle can be used for deleting the task.

Description

HB_IDLEADD() adds a passed codeblock to the list of background tasks that will be evaluated during the idle states. There is no limit for the number of tasks.

Examples

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

Status

Ready

Compliance

Harbour extension similar to FT_ONIDLE() function available in NanForum library.

Platforms

All

Files

source/rtl/idle.c

See Also:

HB_IDLEDEL()
hb_idleState()

HB_IDLEDEL()

Removes the background task from the list of tasks.

Syntax

HB_IDLEDEL(<nHandle>) --> xAction

Arguments

<nHandle> is the identifier of the task returned by the HB_IDLEADD() function.

Returns

<xAction> NIL if invalid handle is passed or a codeblock that was passed to HB_IDLEADD() function

Description

HB_IDLEDEL() removes the action associated with passed identifier from the list of background tasks. The identifer should be the value returned by the previous call of HB_IDLEADD() function.

If specified task is defined then the codeblock is returned otherwise the NIL value is returned.

Examples

```
nTask := HB_IDLEADD( {|| SayTime()} )
INKEY(10)
cbAction := HB_IDLEDEL( nTask )
```

Status

Ready

Compliance

Harbour extension

Platforms

All

Files

source/rtl/idle.c

See Also:

HB_IDLEADD()
hb_idleState()

HB_IdleState()

Evaluates a single background task and calls the garbage collector.

Syntax

```
HB_IDLESTATE()
```

Arguments

Returns

Description

HB_IDLESTATE() requests the garbage collection and executes a single background task defined by the codeblock passed with HB_IDLEADD() function. Every call to this function evaluates a different task in the order of task creation. There are no arguments passed during a codeblock evaluation.

This function can be safely called even if there are no background tasks defined.

Examples

```
nTask1 := HB_IDLEADD( {|| SayTime()} )
nTask2 := HB_IDLEADD( {|| SaveScreen()} )
DO WHILE( !bFinished )
  bFinished :=DOSomethingVeryImportant()
  HB_IdleState()
ENDDO
cbAction := HB_IDLEDEL( nTask1 )
HB_IDLEDEL( nTask2 )
```

Status

Ready

Compliance

Harbour extension similar to FT_IAMIDLE() function available in NanForum library.

Platforms

All

Files

source/rtl/idle.c

See Also:

HB_IDLEADD()
HB_IDLEDEL()

hb_idleState()

Evaluates a single background task and calls the garbage collector.

Syntax

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

Arguments

Returns

Description

hb_idleState() is a C function that requests garbage collection and executes a single background task defined by the codeblock passed with HB_IDLEADD() function. It also releases the CPU time slices for platforms that require it.

Every call for this function evaluates different task in the order of task creation. There are no arguments passed during codeblock evaluation.

This function can be safely called even if there are no background tasks defined.

This function is automatically called from the INKEY() function.

Status

Ready

Platforms

All

Files

source/rtl/idle.c

See Also:

HB_IDLEADD()
HB_IDLEDEL()
hb_idleState()

INKEY()

Extracts the next key code from the Harbour keyboard buffer.

Syntax

INKEY([<nTimeout>] [,<nEvents>]) --> nKey

Arguments

<nTimeout> is an optional timeout value in seconds, with a granularity of 1/10th of a second. If omitted, INKEY() returns immediately. If set to 0, INKEY() waits until an input event occurs. If set to any other value, INKEY() will return either when an input event occurs or when the timeout period has elapsed. If only this parameter is specified and it is not numeric, it will be treated as if it were 0. But if both parameters are specified and this parameter is not numeric, it will be treated as if it were not present.

<nEvents> is an optional mask of input events that are to be enabled. If omitted, defaults to hb_set.HB_SET_EVENTMASK. Valid input masks are in inkey.ch and are explained below. It is recommended that the mask names be used rather than their numeric values, in case the numeric values change in future releases of Harbour. To allow more than one type of input event, simply add the various mask names together.

inkey.ch	Meaning
INKEY_MOVE	Mouse motion events are allowed
INKEY_LDOWN	The mouse left click down event is allowed
INKEY_LUP	The mouse left click up event is allowed
INKEY_RDOWN	The mouse right click down event is allowed
INKEY_RUP	The mouse right click up event is allowed
INKEY_KEYBOARD	All keyboard events are allowed
INKEY_ALL	All mouse and keyboard events are allowed
	HB_INKEY_EXTENDED Extended keyboard codes are used.

hb_set.HB_SET_EVENTMASK.

Returns

-47 to 386 for keyboard events or the range 1001 to 1007 for mouse events. Mouse events and non-printable keyboard events are represented by the K_<event> values listed in inkey.ch. Keyboard event return codes in the range 32 through 127 are equivalent to the printable ASCII character set. Keyboard event return codes in the range 128 through 255 are assumed to be printable, but results may vary based on hardware and nationality. If HB_INKEY_EXTENDED mode is used, then the return value for keyboard events ranges from 1 through 767 and 1077 through 1491, although not all codes are used.

values. If no keyboard modifier was used, then HB_INKEY_NONE is added. The Alt key adds HB_INKEY_ALT, the Ctrl key adds HB_INKEY_CTRL, the Shift key adds HB_INKEY_SHIFT, and enhanced keys (KeyPad+/ and CursorPad keys) add HB_INKEY_ENHANCED. For example, F1 is scan code 59, so if you just press F1, you get key code 315, but Alt+F1 gives 443, Ctrl+F1 gives 571, and Shift+ F1 gives 699. And NumPad+/ gives 1077, 1205, 1333, and 1461. At this time, the only value that can combine with other values is HB_INKEY_ENHANCED (i.e., there are no Alt+Ctl combinations, etc.)

result, if you switch between the normal and extended modes, you need to be aware that some codes get translated into a zero in normal mode (because there is no corresponding code in normal mode) and that these codes get removed from the keyboard input buffer in normal mode and you won't be able to go back and fetch them later in extended mode.

Description

INKEY() can be used to detect input events, such as keypress, mouse movement, or mouse key clicks (up and/or down).

Examples

```
// Wait for the user to press the Esc key
? "Please press the ESC key."
WHILE INKEY( 0.1 ) != K_ESC
END
```

Tests

```
KEYBOARD "AB"; ? INKEY(), INKEY() ==>   65   66
```

Status

Started

Compliance

INKEY() is compliant with the Clipper 5.3 INKEY() function with one exception: The Harbour INKEY() function will raise an argument error if the first parameter is less than or equal to 0 and the second parameter (or the default mask) is not valid, because otherwise INKEY would never return, because it was, in effect, asked to wait forever for no events (Note: In Clipper, this also blocks SET KEY events).

Files

Library is rtl

See Also:

ARRAY()

__KEYBOARD()

DO NOT CALL THIS FUNCTION DIRECTLY!

Syntax

```
KEYBOARD <cString>
CLEAR TYPEAHEAD
```

Arguments

<cString> is the optional string to stuff into the Harbour keyboard buffer after clearing it first. Note: The character ";" is converted to CHR(13) (this is an undocumented CA-Clipper feature).

Returns

Description

Clears the Harbour keyboard typeahead buffer and then inserts an optional string into it.

Examples

```
// Stuff an Enter key into the keyboard buffer
KEYBOARD CHR(13)
// Clear the keyboard buffer
CLEAR TYPEAHEAD
```

Tests

```
KEYBOARD CHR(13); ? INKEY() ==> 13
KEYBOARD ";" ? INKEY() ==> 13
KEYBOARD "HELLO"; CLEAR TYPEAHEAD; ? INKEY() ==> 0
```

Status

Ready

Compliance

__KEYBOARD() is compliant with CA-Clipper 5.3

Files

Library is rtl

See Also:

ARRAY()
KEYBOARD

HB_KEYPUT()

Put an inkey code to the keyboard buffer.

Syntax

HB_KEYPUT(<nInkeyCode>)

Arguments

<nInkeyCode> is the inkey code, which should be inserted into the keyboard buffer.

Returns

Description

Inserts an inkey code to the string buffer. The buffer is *not* cleared in this operation. This function allows to insert such inkey codes which are not in the range of 0 to 255. To insert more than one code, call the function repeatedly. The zero code cannot be inserted.

Examples

```
// Stuff an Alt+PgDn key into the keyboard buffer
HB_KEYPUT( K_ALT_PGDN )
```

Tests

```
HB_KEYPUT( K_ALT_PGDN ) ; ? INKEY() ==> 417
HB_KEYPUT( K_F11 ) ; ? INKEY() ==> -40
```

Status

Ready

Compliance

HB_KEYPUT() is a Harbour extension.

Files

Library is rtl

See Also:

KEYBOARD
ARRAY()
INKEY()

NEXTKEY()

Get the next key code in the buffer without extracting it.

Syntax

NEXTKEY([<nInputMask>]) --> nKey

Arguments

constants. The sole purpose of this argument is to allow switching between using HB_INKEY_EXTENDED key codes and using the normal Clipper-compatible key codes

Returns

<nKey> The value of the next key in the Harbour keyboard buffer.

Description

Returns the value of the next key in the Harbour keyboard buffer without extracting it.

Examples

```
// Use NEXTKEY() with INKEY() to change display characters, or by
// itself to exit the loop, so that the caller can detect the Esc.
LOCAL nKey, cChar := "+"
WHILE TRUE
   ?? cChar
   nKey := NEXTKEY()
   IF nKey == K_ESC
      EXIT
   ELSE
      IF nKey != 0
         cChar := CHR( nKey )
      END IF
   END IF
END WHILE
```

Tests

```
KEYBOARD "AB"; ? NEXTKEY(), NEXTKEY() ==>   65   65
```

Status

Ready

Compliance

NEXTKEY() is compliant with CA-Clipper 5.3, but has been extended for Harbour.

Files

Library is rtl

See Also:

INKEY()
LASTKEY()

LASTKEY()

Get the last key extracted from the keyboard buffer.

Syntax

LASTKEY([<nInputMask>]) --> nKey

Arguments

constants. The sole purpose of this argument is to allow switching between using HB_INKEY_EXTENDED key codes and using the normal Clipper-compatible key codes

Returns

<nKey>   The last key extracted from the keyboard buffer.

Description

Returns the value of the last key exttracted from the Harbour keyboard buffer

Examples

```
// Continue looping unless the ESC key was pressed in MainFunc()
WHILE TRUE
   MainFunc()
   IF LASTKEY() == K_ESC
      EXIT
   ENDIF
END WHILE
```

Tests

```
KEYBOARD "AB"; ? INKEY(), LASTKEY() ==>   65   65
```

Status

Ready

Compliance

LASTKEY() is compliant with CA-Clipper 5.3, but has been extended for Harbour.

Files

Library is rtl

See Also:

INKEY()
LASTKEY()

KEYBOARD

Stuffs the keyboard with a string.

Syntax

KEYBOARD <cString>

Arguments

<cString> String to be processed, one character at a time, by the Harbour keyboard processor

Description

This command stuffs the input buffer with <cString>. The number of characters that can be stuffed into the keyboard buffer is controlled by the SET TYPEAHEAD command and may range from 0 to 32,622, with each character appearing in the ASCII range of 0 to 255. None of the extended keys may be stuffed into the keyboard buffer. Issuing a KEYBOARD " " will clear the keyboard buffer.

Examples

```
// Stuff an Enter key into the keyboard buffer
KEYBOARD CHR(13)
// Clear the keyboard buffer
CLEAR TYPEAHEAD
```

Tests

```
KEYBOARD CHR(13); ? INKEY() ==> 13
KEYBOARD "HELLO"; CLEAR TYPEAHEAD; ? INKEY() ==> 0
```

Status

Ready

Compliance

__KEYBOARD() is compliant with CA-Clipper 5.3

See Also:

ARRAY()
__KEYBOARD()

READKEY()*

Find out which key terminated a READ.

Syntax

```
READKEY() --> nKeyCode
```

Arguments

Returns

READKEY() returns a numeric code representing the key that caused READ to terminate.

Description

READKEY() is used after a READ was terminated to determine the exit key pressed. If the GET buffer was updated during READ, 256 is added to the return code.

READKEY() is a compatibility function so try not to use it. READKEY() is superseded by LASTKEY() which returns the INKEY() code for that key. UPDATED() could be used to find if the GET buffer was changed during the READ.

Status

Ready

Compliance

READKEY() is compliant with CA-Clipper 5.3

Files

Library is rtl

See Also:

@...Get
INKEY()
LASTKEY()
ARRAY()
ARRAY()
ARRAY()

MROW()

Returns the mouse cursor row position.

Syntax

```
MRow() --> nMouseRow
```

Arguments

Returns

<nMouseRow> The mouse cursor row position.

Description

This function returns the current mouse row cursor position. On graphical systems the value represents pixel rows. On character-based systems the value represents character rows as in Clipper.

Examples

```
IF MRow() < 1
   ? "Mouse is on top row!"
ENDIF
```

Status

Ready

Compliance

MROW() is compliant with CA-Clipper 5.3, but has been extended to work on graphical systems as well as character-based systems.

Files

Library is rtl

See Also:

MCOL()

MCOL()

Returns the mouse cursor column position.

Syntax

```
MCol() --> nMouseColumn
```

Arguments

Returns

<nMouseColumn> The mouse cursor column position.

Description

This function returns the column position of the mouse cursor. On graphical systems the value represents pixels. On character-based systems the value represents character columns as in Clipper.

Examples

```
IF MCol() < 1
   ? "Mouse is on left edge!"
ENDIF
```

Status

Ready

Compliance

MROW() is compliant with CA-Clipper 5.3, but has been extended to work on graphical systems as well as character-based systems.

Platforms

All

Files

Library is rtl

See Also:

MROW()

HB_LANGSELECT()

Select a specific nation message module

Syntax

HB_LANGSELECT(<cNewLang>) --> cOldLang

Arguments

<cNewLang> The ID of the country language module The possible values for <cNewLang> is below as is defined in the Lang library,sorted by language.

Returns

<cOldLang> The old language indentifier

Description

This function set a default language module for date/month names, internal warnigs,NatMsg messages and internal errors. When a Lang ID is selected all messages will be output as the current lang selected until another one is selected or the program ends.

Examples

```
REQUEST HB_LANG_PT
REQUEST HB_LANG_RO
REQUEST HB_LANG_ES
FUNCTION MAIN()
HB_LANGSELECT( 'PT' )       // Default language is now Portuguese
? CDOW( DATE() )            // Segunda-feira
? 'Old language id selected is ", HB_LANGSELECT()   // PT
HB_LANGSELECT( 'RO' )       // Default language is now Romanian
? CMONTH( DATE() )          // Mai
? 'Old language id selected is ",HB_LANGSELECT()   // RO
HB_LANGSELECT( 'ES' )       // Default language is now Spanish
? CMONTH( DATE() )          // Mayo
? CDOW( DATE() )            // Lunes
RETURN NIL
```

Tests

See tests/langapi.prg

Status

Ready

Compliance

This function is a Harbour Extension.

Platforms

Dos,Win32,OS/2

Files

Libraty is rtl

See Also:

hb_langName()
NATIONMSG()

HB_LANGNAME()

Return the Name of the Current Language module in USE

Syntax

HB_LANGNAME() --> cLangName

Arguments

Returns

<cLangName> Name of the Current language in use

Description

This function return the current name of the language module in use.

Examples

```
REQUEST HB_LANG_PT
REQUEST HB_LANG_RO
REQUEST HB_LANG_ES
FUNCTION MAIN()
HB_LANGSELECT( 'PT' )       // Default language is now Portuguese
? CDOW( DATE() )            //Segunda-feira
? 'Current language is ", HB_LANGNAME()             // Portuguese
? 'Old language id selected is ", HB_LANGSELECT()   // PT
HB_LANGSELECT( 'RO' )       // Default language is now Romanian
? CMONTH( DATE() )          // Mai
? 'Old language id selected is ",HB_LANGSELECT()    // RO
HB_LANGSELECT( 'ES' )       // Default language is now Spanish
? 'Current language is ",HB_LANGNAME()              // Spanish
? CMONTH( DATE() )          // Mayo
? CDOW( DATE() )            // Lunes
RETURN NIL
```

Tests

See tests/langapi.prg

Status

Ready

Compliance

This function is a Harbour Extension

Platforms

Dos,Win32,OS/2

Files

Library is lang

See Also:

hb_langSelect()
NATIONMSG()

License

Harbour License

Description

THE HARBOUR PROJECT COMPILER LICENSE
====================================

Note: This license applies to most of the files in the source/compiler directory.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA (or visit their web site at http://www.gnu.org/).

THE HARBOUR PROJECT LIBRARY LICENSE
===================================

Note: This license applies to most of the files in the include directory, source directory, and subdirectories.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this software; see the file COPYING. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA (or visit the web site http://www.gnu.org/).

As a special exception, the Harbour Project gives permission for additional uses of the text contained in its release of Harbour.

The exception is that, if you link the Harbour libraries with other files to produce an executable, this does not by itself cause the resulting executable to be covered by the GNU General Public License. Your use of that executable is in no way restricted on account of linking the Harbour library code into it.

This exception does not however invalidate any other reasons why the executable file might be covered by the GNU General Public License.

This exception applies only to the code released by the Harbour Project under the name Harbour. If you copy code from other Harbour Project or Free Software Foundation releases into a copy of Harbour, as the General Public License permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from them.

If you write modifications of your own for Harbour, it is your choice whether to permit this exception to apply to your modifications. If you do not wish that, delete this exception notice.

THE OLD HARBOUR PROJECT LIBRARY LICENSE
=======================================

Note: This license only applies to the following files:
<pre> contrib\libmisc\dates2.c (Only the DateTime() function by Jon Berg) samples\pe\* source\rtl\philes.c source\rtl\binnum.c source\lang\msgsr852.c source\lang\msgpl852.c source\lang\msgpliso.c source\lang\msgplmaz.c source\lang\msgeu.c source\lang\msgcsiso.c source\lang\msgcswin.c

THE HARBOUR PROJECT CONTRIB LICENSE
===================================

See Also:

OVERVIEW

ABS()

Return the absolute value of a number.

Syntax

```
ABS(<nNumber>) --> <nAbsNumber>
```

Arguments

<nNumber> Any number.

Returns

<nAbsNumber> The absolute numeric value.

Description

This function yields the absolute value of the numeric value or expression <nNumber>.

Examples

```
Proc Main()

Local nNumber:=50
Local nNumber1:=27
cls

qout(nNumber-nNumber1)
qout(nNumber1-nNumber)
qout(ABS(nNumber-nNumber1))
qout(ABSnNumber1-nNumber))
qout(ABS( -1 * 345))
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

EXP()

EXP()

Calculates the value of e raised to the passed power.

Syntax

EXP(<nNumber>) --> <nValue>

Arguments

<nNumber> Any real number.

Returns

<nValue> The anti-logarithm of <nNumber>

Description

This function returns the value of e raised to the power of <nNumber>. It is the inverse of LOG().

Examples

```
? EXP(45)
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

LOG()

INT()

Return the integer port of a numeric value.

Syntax

```
INT( <nNumber> ) --> <nIntNumber>
```

Arguments

<nNumber> Any numeric value.

Returns

<nIntNumber> The integer portion of the numeric value.

Description

This function converts a numeric expression to an integer. All decimal digits are truncated. This function does not round a value upward or downward; it merely truncates a number at the decimal point.

Examples

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

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

ROUND()
STRZERO()

LOG()

Returns the natural logarithm of a number.

Syntax

```
LOG( <nNumber> ) --> <nLog>
```

Arguments

<nNumber> Any numeric expression.

Returns

<nExponent> The natural logarithm of <nNumber>.

Description

This function returns the natural logarithm of the number <nNumber>. If <nNumber> is 0 or less than 0, a numeric overflow occurs, which is depicted on the display device as a series of asterisks. This function is the inverse of EXP().

Examples

```
? LOG(632512)
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

EXP()

MAX()

Returns the maximum of two numbers or dates.

Syntax

MAX(<xValue>,<xValue1>) --> <xMax>

Arguments

<xValue>	Any date or numeric value.
<xValue1>	Any date or numeric value (same type as <xValue>).

Returns

<xMax> The larger numeric (or later date) value.

Description

This function returns the larger of the two passed espressions. If <xValue> and <xValue1> are numeric data types, the value returned by this function will be a numeric data type as well and will be the larger of the two numbers passed to it. If <xValue> and <xValue1> are date data types, the return value will be a date data type as well. It will be the later of the two dates passed to it.

Examples

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

Status

Ready

Compliance

This function is Ca-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

MIN()

MIN()

Determines the minumum of two numbers or dates.

Syntax

```
MIN(<xValue>,<xValue1>)  --> <xMin>
```

Arguments

<xValue>	Any date or numeric value.
<xValue1>	Any date or numeric value.

Returns

<xMin>	The smaller numeric (or earlier date) value.

Description

This function returns the smaller of the two passed espressions. <xValue> and <xValue1> must be the same data type. If numeric, the smaller number is returned. If dates, the earlier date is returned.

Examples

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

Status

Ready

Compliance

This function is Ca-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

MAX()

MOD()

Return the modulus of two numbers.

Syntax

MOD(<nNumber>,<nNumber1>) --> <nRemainder>

Arguments

<nNumber>	Numerator in a divisional expression.
<nNumber1>	Denominator in a divisional expression.

Returns

<nRemainder>	The remainder after the division operation.

Description

This functuion returns the remainder of one number divided by another.

Examples

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

Status

Ready

Compliance

This Function is Ca-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

ARRAY()

SQRT()

Calculates the square root of a number.

Syntax

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

Arguments

<nNumber>	Any numeric value.

Returns

<nSqrt>	The square root of <number>.

Description

This function returns the square root of <nNumber>. The precision of this evaluation is based solely on the settings of the SET DECIMAL TO command. Any negative number passed as <nNumber> will always return a 0.

Examples

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

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

ROUND()

ROUND()

Rounds off a numeric expression.

Syntax

```
ROUND( <nNumber>,<nPlace> ) --> <nResult>
```

Arguments

<nNumber>	Any numeric value.
<nPlace>	The number of places to round to.

Returns

<nResult>	The rounded number.

Description

This function rounds off the value of <nNumber> to the number of decimal places specified by <nPlace>. If the value of <nPlace> is a negative number, the function will attempt to round <nNumber> in whole numbers. Numbers from 5 through 9 will be rounded up, all others will be rounded down.

Examples

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

Status

Ready

Compliance

This function is CA-Clipper compliant.

Platforms

All

Files

Library is rtl

See Also:

INT()
STR()
VAL()
SET FIXED

hb_getMathError()

get the last math lib error

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_resetMathError()

reset the math error, i.e. set it to 0

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_isMathHandler()

check if harbour math error handler is available

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_installMathHandler()

add a custom math handler to the math error handler chain

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_deinstallMathHandler()

remove custom math handler from the math error handler chain

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_setMathHandlerStatus()

set the status of a custom math handler in the math error handler chain

Syntax

C Prototype

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

Arguments

can be one of: HB_MATH_HANDLER_STATUS_INACTIVE --> handler is present but not active HB_MATH_HANDLER_STATUS_ACTIVE --> handler is present and active

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

hb_getMathHandlerStatus()

get the status of a custom math handler in the math error handler chain

Syntax

C Prototype

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

Arguments

Returns

Description

Status

Ready
Compliance is not applicable to API calls.

Files

Library is rtl

Platforms

All

MEMOTRAN()

Converts hard and soft carriage returns within strings.

Syntax

```
MEMOTRAN( <cString>, <cHard>, <cSoft> ) --> <cConvertedString>
```

Arguments

<cString> is a string of chars to convert.

<cHard> is the character to replace hard returns with. If not specified defaults to semicolon.

<cSoft> is the character to replace soft returns with. If not specified defaults to single space.

Returns

<cConvertedString> Trasformed string.

Description

Returns a string/memo with carriage return chars converted to specified chars.

Examples

```
? MEMOTRAN( DATA->CNOTES )
```

Tests

```
@ 1, 1 SAY MEMOTRAN( Data->CNOTES )
will display converted string starting on row two, column two of the
current device.
```

Status

Ready

Compliance

MEMOTRAN() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

HARDCR()
STRTRAN()

HARDCR()

Replace all soft carriage returns with hard carriages returns.

Syntax

HARDCR(<cString>) --> <cConvertedString>

Arguments

<cString> is a string of chars to convert.

Returns

<cConvertedString> Trasformed string.

Description

Returns a string/memo with soft carriage return chars converted to hard carriage return chars.

Examples

```
? HARDCR( Data->CNOTES )
```

Tests

```
@ 1, 1 SAY HARDCR( Data->CNOTES )
will display converted string starting on row two, column two of the
current device.
```

Status

Ready

Compliance

HARDCR() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

MEMOTRAN()
STRTRAN()

FIELD

Declares a list of database field names.

Syntax

```
FIELD <xField> [,<xFieldn...>  [in <cDatabase>]
```

Arguments

<xField>	A valid field name
<xFieldn>	Additional field name
<cDatabase>	An valid alias name

Returns

Description

This command declares the names of fields <xField> (and <xFieldn> and following) with an optional alias identifier as <cDatabase> for each. This command allow Harbour to resolve any reference to a field specified in the field listby viewing it as a field when it is not referenced by an alias.If a field is not listed in this list and it is not explicity tagged with an alias indentifier,it may be viewed as a memory variable,which may cause run-time errors.This command has no effect on memory variables or on field reference buried within a macro expression.

Examples

```
Func main
FIELD iD
FIELD Name
USE TESTS NEW
name:="Sales"
Id:=5
USE
Return Nil
```

Tests

See tests/testwarn.prg

Status

Ready

Compliance

This command works exactaly as CA-Clipper.

Platforms

All.

Files

None.

See Also:

MEMVAR
ARRAY()
ARRAY()
ARRAY()

LOCAL

Initializes a local memory variable or array

Syntax

LOCAL <xVar> [:= <xInit>]

Arguments

<xVar>	Name of a memory variable or array.
<xInit>	Value to be assinged to a variable or array

Returns

Description

This command created a LOCAL memory variable or array.The name of either is specified in <xVar>.If more then one variable is being initialized with the LOCAL command,separate each entry with a comma. If a variable or an array is to be assingned a start-up value,that expression may be specified in <xInit> and folling.Is Strong type compile mode is used, the Compiler will check if the value recived matchs the type specified in <xType>.

LOCAL varibles are symbols generated at run time and are resolved at compile time.The visibility and life span of a LOCAL variable or array is limited to the function or procedure in which it is defined.

No macro expansions are allowed in the LOCAL declaration statement.

No Harbour command other then FUNCTION,PROCEDURE,PUBLIC,PRIVATE, PARAMETERS,MEMVAR,STATIC and FIELD,may precede the LOCAL command.

LOCAL array reference may not be initialized(i.e.,assigned values) on the same command line as the LOCAL command statement.This can be done later in the program.

LOCAL variables and arrays are not affected by the RELEASE command.

Examples

```
Function Main2()
Local n , lVar

n := IIF( lVar, 'A', 3 )
n := 2
n := 'a'
n := seconds() + 2
n := int( seconds() + 2 )
Return( NIL )
```

Tests

See Tests/testwarn.prg for more examples

Status

Ready

Compliance

This command works exactaly as CA-Clipper.

Platforms

All

Files

None

See Also:

FIELD
ARRAY()

ARRAY()
ARRAY()
MEMVAR

MEMVAR

Declares private and public variables and arrays.

Syntax

```
MEMVAR <xVar>
```

Arguments

<xVar> Memory variable Name

Returns

Description

This command tells the compiler to resolve any reference to a memory variable designated within this list s if it possessed an explicit memory variable alias with either the M-> or MEMVAR-> prefix.Only those memory variables that do not contain any such explicit are affected by this command.Those memory variabls within macro expansions are not affected by this command.

The MEMVAR declaration must apear before any executable commands;it is similat to the LOCAL,STATIC,FIELD,PARAMETERS,FUNCTION, and PROCEDURE commands statements.

Examples

```
MEMVAR y As Numeric
Function Main2()
Local n , lVar

n := IIF( lVar, 'A', 3 )
n := 2
n := 'a'
n := seconds() + 2
n := int( seconds() + 2 )
y := n
? y
Return( NIL )
```

Tests

See Tests/testwarn.prg for more examples

Status

Ready

Compliance

This command works exactaly as CA-Clipper.

Platforms

All

Files

None.

See Also:

LOCAL
ARRAY()
FIELD
ARRAY()
ARRAY()

ACHOICE()

Allows selection of an element from an array

Syntax

ACHOICE(<nTop>, <nLeft>, <nBottom>, <nRight>, <acMenuItems>, [<alSelableItems> | <lSelableItems>], [<cUserFunction> | <bUserBlock>], [<nInitialItem>], [<nWindowRow>]) --> nPosition

Arguments

<nTop> - topmost row used to display array (default 0)

<nLeft> - leftmost row used to display array (default 0)

<nBottom> - bottommost row used to display array (default MAXROW())

<nRight> - rightmost row used to display array (default MAXCOL())

<acMenuItems> - the character array of items from which to select

<alSelableItems> - an array of items, either logical or character, which is used to determine if a particular item may be selected. If the type of a given item is character, it is macro evaluated, and the result is expected to be a logical. A value of .T. means that the item may be selected, .F. that it may not. (See next argument: lSelectableItems)

<lSelableItems> - a logical value which is used to apply to all items in acMenuItems. If .T., all items may be selected; if .F., none may be selected. (See previous argument: alSelectableItems) Default .T.

<cUserFunction> - the name of a function to be called which may affect special processing of keystrokes. It is specified without parentheses or parameters. When it is called, it will be supplied with the parameters: nMode, nCurElement, and nRowPos. Default NIL.

<bUserBlock> - a codeblock to be called which may affect special processing of keystrokes. It should be specified in the form {|nMode, nCurElemenet, nRowPos| ; MyFunc(nMode, nCurElemenet, nRowPos) }. Default NIL.

<nInitialItem> - the number of the element to be highlighted as the current item when the array is initially displayed. 1 origin. Default 1.

<nWindowRow> - the number of the window row on which the initial item is to be displayed. 0 origin. Default 0.

Returns

<nPosition> - the number of the item to be selected, or 0 if the selection was aborted.

Description

Allows selection of an element from an array. Please see standard Clipper documentation for ACHOICE for additional detail.

Examples

```
aItems := { "One", "Two", "Three" }
nChoice := ACHOICE( 10, 10, 20, 20, aItems )
IF nChoice == 0
    ? "You did not choose an item"
ELSE
    ? "You chose element " + LTRIM( STR( nChoice ) )
    ?? " which has a value of " + aItems[ nChoice ]
ENDIF
```

Files

Library is rtl

See Also:

MENU TO

__AtPrompt()

Display a menu item on screen and define a message

Syntax

```
__AtPrompt( <nRow>, <nCol>, <cPrompt>, [<xMsg>] ) --> .F.
```

Arguments

<nRow> is the row number to display the menu <cPrompt>. Value could range from zero to MAXROW().

<nCol> is the column number to display the menu <cPrompt>. Value could range from zero to MAXCOL().

<cPrompt> is the menu item character string to display.

<xMsg> define a message to display each time this menu item is highlighted. <xMsg> could be a character string or code block that is evaluated to a character string. If <xMsg> is not specified or got the wrong type, an empty string ("") would be used.

Returns

__AtPrompt() always return .F.

Description

With __AtPrompt() you define and display a menu item, each call to __AtPrompt() add another item to the menu, to start the menu itself you should call the __MenuTo() function (MENU TO command). You can define any row and column combination and they will be displayed at the order of definition. After each call to __AtPrompt(), the cursor is placed one column to the right of the last text displayed, and ROW() and COL() are updated.

@...PROMPT command is preprocessed into __AtPrompt() function during compile time.

Examples

```
// display a two line menu with status line at the bottom
// let the user select favorite day
SET MESSAGE TO 24 CENTER
@ 10, 2 PROMPT "Sunday" MESSAGE "This is the 1st item"
@ 11, 2 PROMPT "Monday" MESSAGE "Now we're on the 2nd item"
MENU TO nChoice
DO CASE
   CASE nChoice == 0           // user press Esc key
        QUIT
   CASE nChoice == 1           // user select 1st menu item
        ? "Guess you don't like Mondays"
   CASE nChoice == 2           // user select 2nd menu item
        ? "Just another day for some"
ENDCASE
```

Status

Ready

Compliance

CA-Clipper array is limited to 4096 items, and therefor 4096 menu items are the maximum that could be defined per one menu, Harbour does not have this limit (not that you'll ever need that).

Files

Library is rtl

See Also:

ACHOICE()
MENU TO
SET MESSAGE
SET INTENSITY

SET WRAP
   MenuTo()

@...PROMPT

Display a menu item on screen and define a message

Syntax

@ <nRow>, <nCol> PROMPT <cPrompt> [MESSAGE <xMsg>]

Arguments

<nRow> is the row number to display the menu <cPrompt>. Value could range from zero to MAXROW().

<nCol> is the column number to display the menu <cPrompt>. Value could range from zero to MAXCOL().

<cPrompt> is the menu item character string to display.

<xMsg> define a message to display each time this menu item is highlighted. <xMsg> could be a character string or code block that is evaluated to a character string. If <xMsg> is not specified or got the wrong type, an empty string ("") would be used.

Returns

@...**Prompt** always return .F.

Description

With @...Prompt you define and display a menu item, each call to @...Prompt add another item to the menu, to start the menu itself you should call the __MenuTo() function (MENU TO command). You can define any row and column combination and they will be displayed at the order of definition. After each call to @...Prompt, the cursor is placed one column to the right of the last text displayed, and ROW() and COL() are updated.

@...PROMPT command is preprocessed into __AtPrompt() function during compile time.

Examples

```
// display a two line menu with status line at the bottom
// let the user select favorite day
SET MESSAGE TO 24 CENTER
@ 10, 2 PROMPT "Sunday" MESSAGE "This is the 1st item"
@ 11, 2 PROMPT "Monday" MESSAGE "Now we're on the 2nd item"
MENU TO nChoice
DO CASE
   CASE nChoice == 0           // user press Esc key
        QUIT
   CASE nChoice == 1           // user select 1st menu item
        ? "Guess you don't like Mondays"
   CASE nChoice == 2           // user select 2nd menu item
        ? "Just another day for some"
ENDCASE
```

Status

Ready

Compliance

CA-Clipper array is limited to 4096 items, and therefor 4096 menu items are the maximum that could be defined per one menu, Harbour does not have this limit (not that you'll ever need that).

See Also:

ACHOICE()
MENU TO
SET MESSAGE
SET INTENSITY
SET WRAP
__MenuTo()

__MenuTo()

Invoked a menu defined by set of @...PROMPT

Syntax

__MenuTo(<bBlock>, <cVariable>) --> nChoice

Arguments

<bBlock> is a set/get code block for variable named <cVariable>.

<cVariable> is a character string that contain the name of the variable to hold the menu choices, if this variable does not exist a PRIVATE variable with the name <cVariable> would be created to hold the result.

Returns

__MenuTo() return the number of select menu item, or 0 if there was no item to select from or if the user pressed the Esc key.

Description

__MenuTo() invoked the menu define by previous __AtPrompt() call and display a highlight bar that the user can move to select an option from the menu. If <cVariable> does not exist or not visible, a PRIVATE variable named <cVariable> is created and hold the current menu selection. If there is a variable named <cVariable>, its value is used to select the first highlighted item.

Menu prompts and messages are displayed in current Standard color, highlighted bar is displayed using current Enhanced color.

Pressing the arrow keys move the highlighted bar. When a menu item is highlighted the message associated with it is displayed on the line specified with SET MESSAGE. If SET WRAP is ON and the user press UP arrow while on the first selection the last menu item is highlighted, if the user press Down arrow while on the last item, the first item is highlighted.

Following are active keys that handled by __MenuTo():
----

key	Meaning
Up	Move to previous item
Down	Move to next item
Left	Move to previous item
Right	Move to next item
Home	Move to the first item
End	Move to the last item
Page-Up	Select menu item, return position
Page-Down	Select menu item, return position
Enter	Select menu item, return position
Esc	Abort selection, return 0
First letter	Select next menu with the same first letter,
|	return this item position.

upon exit the cursor is placed at MAXROW()-1, 0 __MenuTo() can be nested without loosing the previous prompts.

MENU TO command is preprocessed into __MenuTo() function during compile time.

Examples

```
// display menu item on each screen corner and let user select one
CLS
SET MESSAGE TO MAXROW()/2 CENTER
SET WRAP ON
```

```
@ 0,          0            PROMPT "1. Upper left"   MESSAGE " One "
@ 0,          MAXCOL()-16 PROMPT "2. Upper right"  MESSAGE " Two "
@ MAXROW()-1,MAXCOL()-16 PROMPT "3. Bottom right" MESSAGE "Three"
@ MAXROW()-1,0            PROMPT "4. Bottom left"  MESSAGE "Four "
MENU TO nChoice
SETPOS ( MAXROW()/2, MAXCOL()/2 - 10 )
if nChoice == 0
   ?? "Esc was pressed"
else
   ?? "Selected option is", nChoice
endif
```

Status

Ready

Compliance

This command is CA-Clipper compliant

Files

Library is rtl

See Also:

@...PROMPT
ACHOICE()
SET MESSAGE
SET INTENSITY
SET WRAP
__AtPrompt()

MENU TO

Invoked a menu defined by set of @...PROMPT

Syntax

MENU TO <cVariable>

Arguments

<cVariable> is a character string that contain the name of the variable to hold the menu choices, if this variable does not exist a PRIVATE variable with the name <cVariable> would be created to hold the result.

Returns

from or if the user pressed the Esc key.

Description

Menu To() invoked the menu define by previous __AtPrompt() call and display a highlight bar that the user can move to select an option from the menu. If <cVariable> does not exist or not visible, a PRIVATE variable named <cVariable> is created and hold the current menu selection. If there is a variable named <cVariable>, its value is used to select the first highlighted item.

Menu prompts and messages are displayed in current Standard color, highlighted bar is displayed using current Enhanced color.

Pressing the arrow keys move the highlighted bar. When a menu item is highlighted the message associated with it is displayed on the line specified with SET MESSAGE. If SET WRAP is ON and the user press UP arrow while on the first selection the last menu item is highlighted, if the user press Down arrow while on the last item, the first item is highlighted.

Following are active keys that handled by Menu To:
----

key	Meaning
Up	- Move to previous item
Down	- Move to next item
Left	- Move to previous item
Right	- Move to next item
Home	- Move to the first item
End	- Move to the last item
Page-Up	- Select menu item, return position
Page-Down	- Select menu item, return position
Enter	- Select menu item, return position
Esc	- Abort selection, return 0
First letter	- Select next menu with the same first letter,
|	return this item position.

upon exit the cursor is placed at MAXROW()-1, 0 Menu To can be nested without loosing the previous prompts.

MENU TO command is preprocessed into __MenuTo() function during compile time.

Examples

```
// display menu item on each screen corner and let user select one
CLS
SET MESSAGE TO MAXROW()/2 CENTER
SET WRAP ON
@ 0,         0           PROMPT "1. Upper left"   MESSAGE " One "
@ 0,         MAXCOL()-16 PROMPT "2. Upper right"  MESSAGE " Two "
@ MAXROW()-1,MAXCOL()-16 PROMPT "3. Bottom right" MESSAGE "Three"
@ MAXROW()-1,0           PROMPT "4. Bottom left"  MESSAGE "Four "
MENU TO nChoice
SETPOS ( MAXROW()/2, MAXCOL()/2 - 10 )
if nChoice == 0
   ?? "Esc was pressed"
else
   ?? "Selected option is", nChoice
endif
```

Status

Ready

Compliance

This command is CA Clipper compliant

See Also:

@...PROMPT
ACHOICE()
SET MESSAGE
SET INTENSITY
SET WRAP
__AtPrompt()

OS()

Return the current operating system.

Syntax

```
OS() --> <cOperatingSystem>
```

Returns

<cOperatinSystem> The Current operating system.

Description

This function will return the current operating system.

Examples

```
? OS()
```

Status

Ready

Compliance

This function is CA-Clipper compatible.

Platforms

All

Files

source/rtl/version.c

VERSION()

Returns the HARBOUR Version or the Harbour/Compiler Version.

Syntax

VERSION() --> <cReturn>

Arguments

Returns

<cReturn> String containing the Harbour Version

Description

This function returns the current Harbour Version.

Examples

```
? QOUT(VERSION())
"Harbour Terminal: Standard stream console"
```

Status

Started

Compliance

This function is Ca-Clipper compatible.

Platforms

All

Files

source/rtl/version.c Library is rtl

See Also:

OS()

GETENV()

Obtains system environmental settings.

Syntax

GETENV(<cEnviroment>, <cDefaultValue>) --> <cReturn>

Arguments

<cEnviroment> Enviromental variable to obtain.

<cDefaultValue> Optional value to return if <cEnvironment> is not found.

Returns

<cReturn>   Value of the Environment Variable.

Description

This function yields a string that is the value of the environment variable <cEnviroment>, which is stored at the system level with the Set command. If no environment variable can be found, the value of the function will be <cDefaultValue> if it is passed, else an empty string.

Examples

```
? QOUT(GETENV('PATH'))
? QOUT(GETENV('CONFIG'))
? QOUT(GETENV('HARBOURCMD', '-n -l -es2'))
```

Status

Ready

Compliance

This command is Ca-Clipper compliant. The <cDefaultValue> parameter is a Harbour extension.

Platforms

All

Files

source/rtl/gete.c Library is rtl

__RUN()

Run an external program.

Syntax

```
__RUN( <cCommand> )
```

Arguments

<cCommand> Command to execute.

Description

This command runs an external program. Please make sure that you have enough free memory to be able to run the external program. Do not use it to run Terminate and Stay Resident programs (in case of DOS) since that causes several problems.

Note: This function is what the RUN command preprocesses into. It is considered bad form to use this function directly. Use the RUN command instead.

Examples

```
__Run( "edit " + cMyTextFile )    // Runs an external editor
__Run( "command" )                // Gives a DOS shell (DOS only)
```

Status

Ready

Compliance

This function is Ca-Clipper compliant.

Platforms

All

Files

source/rtl/run.c Library is rtl

See Also:

RUN

TONE()

Sound a tone with a specified frequency and duration.

Syntax

TONE(<nFrequency>, <nDuration>) --> NIL

Arguments

<nFrequency> A non-negative numeric value that specifies the frequency of the tone in hertz.

<nDuration> A positive numeric value which specifies the duration of the tone in 1/18 of a second units.

Returns

TONE() always returns NIL.

Description

TONE() is a sound function that could be used to irritate the end user, his or her dog, and the surrounding neighborhood. The frequency is clamped to the range 0 to 32767 Hz.

Examples

```
If lOk   // Good Sound
   TONE(  500, 1 )
   TONE( 4000, 1 )
   TONE( 2500, 1 )
Else     // Bad Sound
   TONE(  300, 1 )
   TONE(  499, 5 )
   TONE(  700, 5 )
EndIf
```

Tests

```
TONE( 800, 1 )                         // same as ? CHR(7)
TONE( 32000, 200 )                     // any dogs around yet?
TONE( 130.80, 1 )                      // musical note - C
TONE( 400, 0 )                         // short beep
TONE( 700 )                            // short beep
TONE( 10, 18.2 )                       // 1 second delay
TONE( -1 )                             // 1/18.2 second delay
TONE( )                                // 1/18.2 second delay
```

Tests

Status

Started

Compliance

TONE() works exactly like CA-Clipper's TONE().

Platforms

All

Files

Library is rtl

See Also:

CHR()
SET BELL

RUN

Run an external program.

Syntax

```
RUN  <cCommand>
```

Arguments

<cCommand> Command to execute.

Description

This command runs an external program. Please make sure that you have enough free memory to be able to run the external program. Do not use it to run Terminate and Stay Resident programs (in case of DOS) since that causes several problems.

Examples

```
Run  "edit " + cMyTextFile      // Runs an external editor
Run  "command"                  // Gives a DOS shell (DOS only)
```

Status

Ready

Compliance

This command is Ca-Clipper compliant.

Platforms

All

Files

source/rtl/run.c Library is rtl

See Also:

RUN

ISAFFIRM()

Checks if passed char is an affirmation char

Syntax

ISAFFIRM(<cChar>) --> <lTrueOrFalse>

Arguments

<cChar> is a char or string of chars

Returns

<lTrueOrFalse> True if passed char is an affirmation char,otherwise false

Description

This function is used to check if a user's input is true or not according to the msgxxx module used.

Examples

```
// Wait until user enters Y
DO WHILE !ISAFFIRM( cYesNo )
  ACCEPT "Sure: " TO cYesNo
END DO
```

Status

Ready

Compliance

ISAFFIRM() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ISNEGATIVE()
NATIONMSG()

ISNEGATIVE()

Checks if passed char is a negation char.

Syntax

ISNEGATIVE(<cChar>) --> <lTrueOrFalse>

Arguments

<cChar> is a char or string of chars

Returns

<lTrueOrFalse> True if passed char is a negation char, otherwise false.

Description

This function is used to check if a user's input is true or not according to the msgxxx module used.

Examples

```
// Wait until user enters N
DO WHILE !ISNEGATIVE( cYesNo )
  ACCEPT "Sure: " TO cYesNo
END DO
```

Status

Ready

Compliance

ISNEGATIVE() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ISAFFIRM()
NATIONMSG()

NATIONMSG()

Returns international strings messages.

Syntax

NATIONMSG(<nMsg>) --> <cMessage>

Arguments

<nMsg> is the message number you want to get.

Returns

<cMessage> If <nMsg> is a valid message selector, returns the message. If <nMsg> is nil returns "Invalid Argument", and if <nMsg> is any other type it returns an empty string.

Description

NATIONMSG() returns international message descriptions.

Examples

```
// Displays "Sure Y/N: "  and waits until user enters Y
// Y/N is the string for NATIONMSG( 12 ) with default natmsg module.
DO WHILE !ISAFFIRM( cYesNo )
  ACCEPT "Sure " + NATIONMSG( 12 ) + ": " TO cYesNo
END DO
```

Status

Clipper

Compliance

NATIONMSG() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ISAFFIRM()
ISNEGATIVE()

__objHasData()

Determine whether a symbol exist in object as DATA

Syntax

__objHasData(<oObject>, <cSymbol>) --> lExist

Arguments

<oObject>	is an object to scan.
<cSymbol>	is the name of the symbol to look for.

Returns

__objHasData() return .T. if the given <cSymbol> exist as DATA (instance variable) in object <oObject), .F. if it does not exist.

Description

__objHasData() is a low level class support function that let you find out if a symbol is an instance variable in a given object.

Examples

```
oB := TBrowseNew( 0, 0, 24, 79 )
? __objHasData( oB, "nLeft" )      // this should return .T.
? __objHasData( oB, "lBugFree" )   // hopefully this should be .F.
? __objHasData( oB, "Left" )       // .F. since this is a METHOD
```

Status

Ready

Compliance

__objHasData() is a Harbour extension.

Files

Library is rtl

See Also:

__objGetMethodList()
__objGetMsgList()
__objHasMethod()

__objHasMethod()

Determine whether a symbol exist in object as METHOD

Syntax

__objHasMethod(<oObject>, <cSymbol>) --> lExist

Arguments

<oObject> is an object to scan.

<cSymbol> is the name of the symbol to look for.

Returns

__objHasMethod() return .T. if the given <cSymbol> exist as METHOD (class function) in object <oObject), .F. if it does not exist.

Description

__objHasMethod() is a low level class support function that let you find out if a symbol is a class function in a given object.

Examples

```
oB := TBrowseNew( 0, 0, 24, 79 )
? __objHasMethod( oB, "nLeft" )      // .F. since this is a DATA
? __objHasMethod( oB, "FixBugs" )    // hopefully this should be .F.
? __objHasMethod( oB, "Left" )       // this should return .T.
```

Status

Ready

Compliance

__objHasMethod() is a Harbour extension.

Files

Library is rtl

See Also:

__objGetMethodList()
__objGetMsgList()
__objHasData()

__objGetMsgList()

Return names of all DATA or METHOD for a given object

Syntax

__objGetMsgList(<oObject>, [<lData>], [nClassType]) --> aNames

Arguments

<oObject> is an object to scan.

<lData> is an optional logical value that specifies the information to return. A value of .T. instruct the function to return list of all DATA names, .F. return list of all METHOD names. Default value is .T.

<nClassType> is on optional numeric code for selecting which class type to return. Default value is HB_MSGLISTALL, returning the whole list.

Returns

__objGetMsgList() return an array of character stings with all DATA names or all METHOD names for a given object. __objGetMsgList() would return an empty array {} if the given object does not contain the requested information.

Description

__objGetMsgList() is a low level class support function that let you find all instance variable or method names for a given object.

If specified, the following table shoes the values for <nClassType> that allow you to distinguish between DATA and CLASSDATA:

hboo.ch	Value Meaning
HB_MSGLISTALL	0 All types
HB_MSGLISTCLASS 1	CLASSDATA only
HB_MSGLISTPURE	2 DATA only

DATA are instance variable usable within each object from a class, where each object has its own DATAs.

CLASSDATA are shared by all objects from a Class, so the changed value within Object1 will be reflected when accessing the CLASSDATA from Object2.

Examples

```
// show information about TBrowse class
oB := TBrowseNew( 0, 0, 24, 79 )
aData      := __objGetMsgList( oB, .T. )
aClassData := __objGetMsgList( oB, .T., HB_MSGLISTCLASS )
aMethod    := __objGetMsgList( oB, .F. )
FOR i = 1 to len ( aData )
    ? "DATA name:", aData[ i ]
NEXT
FOR i = 1 to len ( aClassData )
    ? "CLASSDATA name:", aClassData[ i ]
NEXT
FOR i = 1 to len ( aMethod )
    ? "METHOD name:", aMethod[ i ]
NEXT
```

Status

Ready

Compliance

__objGetMsgList() is a Harbour extension.

Files

Header file is hboo.ch Library is rtl

See Also:

__objGetMethodList()
__objGetValueList()
__objHasData()
__objHasMethod()

__objGetMethodList()

Return names of all METHOD for a given object

Syntax

__objGetMethodList(<oObject>) --> aMethodNames

Arguments

<oObject> is an object to scan.

Returns

__objGetMethodList() return an array of character stings with all METHOD names for a given object. __objGetMethodList() would return an empty array {} if the given object does not contain any METHOD.

Description

__objGetMethodList() is a low level class support function that let you find all class functions names for a given object. It is equivalent to __objGetMsgList(oObject, .F.).

Examples

```
// show information about TBrowse class
oB := TBrowseNew( 0, 0, 24, 79 )
aMethod := __objGetMethodList( oB )
FOR i = 1 to len ( aMethod )
    ? "METHOD name:", aMethod[ i ]
NEXT
```

Status

Ready

Compliance

__objGetMethodList() is a Harbour extension.

Files

Library is rtl

See Also:

__objGetMsgList()
__objGetValueList()
__objHasData()
__objHasMethod()

__objGetValueList()

Return an array of DATA names and values for a given object

Syntax

__objGetValueList(<oObject>, [<aExcept>]) --> aData

Arguments

<oObject> is an object to scan.

<aExcept> is an optional array with DATA names you want to exclude from the scan.

Returns

__objGetValueList() return a 2D array that contain pairs of a DATA symbol name and the value of DATA. __objGetValueList() would return an empty array {} if the given object does not contain the requested information.

Description

__objGetValueList() is a low level class support function that return an array with DATA names and value, each array element is a pair of: aData[i, HB_OO_DATA_SYMBOL] contain the symbol name aData[i, HB_OO_DATA_VALUE] contain the value of DATA

Examples

```
// show information about TBrowse class
oB := TBrowseNew( 0, 0, 24, 79 )
aData := __objGetValueList( oB )
FOR i = 1 to len ( aData )
    ? "DATA name:", aData[ i, HB_OO_DATA_SYMBOL ], ;
      "    value=", aData[ i, HB_OO_DATA_VALUE  ]
NEXT
```

Status

Ready

Compliance

__objGetValueList() is a Harbour extension.

Files

Header file is hboo.ch Library is rtl

See Also:

__objGetMethodList()
__objGetMsgList()
__objHasData()
__objHasMethod()
__ObjSetValueList()

__ObjSetValueList()

Set object with an array of DATA names and values

Syntax

__ObjSetValueList(<oObject>, <aData>) --> oObject

Arguments

<oObject> is an object to set.

<aData> is a 2D array with a pair of instance variables and values for setting those variable.

Returns

__ObjSetValueList() return a reference to <oObject>.

Description

__ObjSetValueList() is a low level class support function that let you set a group of instance variables with values. each array element in <aData> is a pair of: aData[i, HB_OO_DATA_SYMBOL] which contain the variable name to set aData[i, HB_OO_DATA_VALUE] contain the new variable value.

Examples

```
// set some TBrowse instance variable
oB := TBrowse():New()
aData := array( 4, 2 )
aData[ 1, HB_OO_DATA_SYMBOL ] = "nTop"
aData[ 1, HB_OO_DATA_VALUE  ] = 1
aData[ 2, HB_OO_DATA_SYMBOL ] = "nLeft"
aData[ 2, HB_OO_DATA_VALUE  ] = 10
aData[ 3, HB_OO_DATA_SYMBOL ] = "nBottom"
aData[ 3, HB_OO_DATA_VALUE  ] = 20
aData[ 4, HB_OO_DATA_SYMBOL ] = "nRight"
aData[ 4, HB_OO_DATA_VALUE  ] = 70
__ObjSetValueList( oB, aData )
? oB:nTop      // 1
? oB:nLeft     // 10
? oB:nBottom   // 20
? oB:nRight    // 70
```

Status

Ready

Compliance

__ObjSetValueList() is a Harbour extension.

Files

Header file is hboo.ch Library is rtl

See Also:

__objGetValueList()

__objAddMethod()

Add a METHOD to an already existing class

Syntax

__objAddMethod(<oObject>, <cMethodName>, <nFuncPtr>) --> oObject

Arguments

<oObject> is the object to work on.

<cMethodName> is the symbol name of the new METHOD to add.

<nFuncPtr> is a pointer to a function to associate with the method.

Returns

__objAddMethod() return a reference to <oObject>.

Description

__objAddMethod() is a low level class support function that add a new METHOD to an object. <oObject> is unchanged if a symbol with the name <cMethodName> already exist in <oObject>.

Note that <nFuncPtr> is a special pointer to a function that was created using the @ operator, see example below.

Examples

```
// create a new THappy class and add a Smile method
oHappy := HBClass():New( "THappy" )
__objAddMethod( oHappy, "Smile", @MySmile() )
? oHappy:Smile( 1 )       // :)
? oHappy:Smile( 2 )       // ;)
? oHappy:Smile( 3 )       // *SMILE*

STATIC FUNCTION MySmile( nType )
LOCAL cSmile
DO CASE
   CASE nType == 1
        cSmile := ":)"
   CASE nType == 2
        cSmile := ";)"
   CASE nType == 3
        cSmile := "*SMILE*"
ENDCASE
RETURN cSmile
```

Status

Ready

Compliance

__objAddMethod() is a Harbour extension.

Files

Library is rtl

See Also:

- __objAddInline()
- __objAddData()
- __objDelMethod()
- __objGetMethodList()
- __objGetMsgList()
- __objHasMethod()
- __objModMethod()

__objAddInline()

Add an INLINE to an already existing class

Syntax

__objAddInline(<oObject>, <cInlineName>, <bInline>) --> oObject

Arguments

<oObject> is the object to work on.

<cInlineName> is the symbol name of the new INLINE to add.

<bInline> is a code block to associate with the INLINE method.

Returns

__objAddInline() return a reference to <oObject>.

Description

__objAddInline() is a low level class support function that add a new INLINE method to an object. <oObject> is unchanged if a symbol with the name <cInlineName> already exist in <oObject>.

Examples

```
// create a new THappy class and add a Smile INLINE method
oHappy  := HBClass():New( "THappy" )
bInline := { | nType | { ":)", ";)", "*SMILE*" }[ nType ] }
__objAddInline( oHappy, "Smile", bInline )
? oHappy:Smile( 1 )       // :)
? oHappy:Smile( 2 )       // ;)
? oHappy:Smile( 3 )       // *SMILE*
```

Status

Ready

Compliance

__objAddInline() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddData()
__objAddMethod()
__objDelInline()
__objGetMethodList()
__objGetMsgList()
ARRAY()
__objModInline()

__objAddData()

Add a DATA to an already existing class

Syntax

__objAddData(<oObject>, <cDataName>) --> oObject

Arguments

<oObject> is the object to work on.

<cDataName> is the symbol name of the new DATA to add.

Returns

__objAddData() return a reference to <oObject>.

Description

__objAddData() is a low level class support function that add a new DATA to an object. <oObject> is unchanged if a symbol with the name <cDataName> already exist in <oObject>.

Examples

```
// create a new THappy class and add a lHappy DATA
oHappy  := HBClass():New( "THappy" )
__objAddData( oHappy, "lHappy" )
oHappy:lHappy := .T.
IF oHappy:lHappy
   ? "Happy, Happy, Joy, Joy !!!"
ELSE
   ? ":(..."
ENDIF
```

Status

Ready

Compliance

__objAddData() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddInline()
__objAddMethod()
__objDelData()
__objGetMsgList()
__objGetValueList()
__objHasData()
__ObjSetValueList()

__objModMethod()

Modify (replace) a METHOD in an already existing class

Syntax

__objModMethod(<oObject>, <cMethodName>, <nFuncPtr>) --> oObject

Arguments

<oObject> is the object to work on.

<cMethodName> is the symbol name of the METHOD to modify.

<nFuncPtr> is a pointer to a new function to associate with the method.

Returns

__objModMethod() return a reference to <oObject>.

Description

__objModMethod() is a low level class support function that modify a METHOD in an object and replace it with a new function. <oObject> is unchanged if a symbol with the name <cMethodName> does not exist in <oObject>. __objModMethod() is used in inheritance mechanism.

Note that <nFuncPtr> is a special pointer to a function that was created using the @ operator, see example below.

Examples

```
// create a new THappy class and add a Smile method
oHappy := HBClass():New( "THappy" )
__objAddMethod( oHappy, "Smile", @MySmile() )
? oHappy:Smile( 1 )       // :)
? oHappy:Smile( 2 )       // ;)
// replace Smile method with a new function
__objAddMethod( oHappy, "Smile", @YourSmile() )
? oHappy:Smile( 1 )       // *SMILE*
? oHappy:Smile( 2 )       // *WINK*

STATIC FUNCTION MySmile( nType )
LOCAL cSmile
DO CASE
   CASE nType == 1
        cSmile := ":)"
   CASE nType == 2
        cSmile := ";)"
ENDCASE
RETURN cSmile

STATIC FUNCTION YourSmile( nType )
LOCAL cSmile
DO CASE
   CASE nType == 1
        cSmile := "*SMILE*"
   CASE nType == 2
        cSmile := "*WINK*"
ENDCASE
RETURN cSmile
```

Status

Ready

Compliance

__objModMethod() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddMethod()
__objDelMethod()
__objGetMethodList()
__objGetMsgList()
__objHasMethod()

__objModInline()

Modify (replace) an INLINE method in an already existing class

Syntax

__objModInline(<oObject>, <cInlineName>, <bInline>) --> oObject

Arguments

<oObject> is the object to work on.

<cInlineName> is the symbol name of the INLINE method to modify.

<bInline> is a new code block to associate with the INLINE method.

Returns

__objModInline() return a reference to <oObject>.

Description

__objModInline() is a low level class support function that modify an INLINE method in an object and replace it with a new code block. <oObject> is unchanged if a symbol with the name <cInlineName> does not exist in <oObject>. __objModInline() is used in inheritance mechanism.

Examples

```
// create a new THappy class and add a Smile INLINE method
oHappy  := HBClass():New( "THappy" )
bMyInline   := { | nType | { ":)", ";)" }[ nType ] }
bYourInline := { | nType | { "*SMILE*", "*WINK*" }[ nType ] }
__objAddInline( oHappy, "Smile", bMyInline )
? oHappy:Smile( 1 )       // :)
? oHappy:Smile( 2 )       // ;)
// replace Smile inline method with a new code block
__objModInline( oHappy, "Smile", bYourInline )
? oHappy:Smile( 1 )       // *SMILE*
? oHappy:Smile( 2 )       // *WINK*
```

Status

Ready

Compliance

__objModInline() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddInline()
__objDelInline()
__objGetMethodList()
__objGetMsgList()
__objHasMethod()

__objDelMethod()

Delete a METHOD from class

Syntax

__objDelMethod(<oObject>, <cSymbol>) --> oObject

Arguments

<oObject> is the object to work on.

<cSymbol> is the symbol name of METHOD or INLINE method to be deleted (removed) from the object.

Returns

__objDelMethod() return a reference to <oObject>.

Description

__objDelMethod() is a low level class support function that delete (remove) a METHOD or an INLINE method from an object. <oObject> is unchanged if a symbol with the name <cSymbol> does not exist in <oObject>.

__objDelInline() is exactly the same as __objDelMethod().

Examples

```
// create a new THappy class and add a Smile method
oHappy := HBClass():New( "THappy" )
__objAddMethod( oHappy, "Smile", @MySmile() )
? __objHasMethod( oHappy, "Smile" )    // .T.
// remove Smile method
__objDelMethod( oHappy, "Smile" )
? __objHasMethod( oHappy, "Smile" )    // .F.

STATIC FUNCTION MySmile( nType )
LOCAL cSmile
DO CASE
   CASE nType == 1
        cSmile := ":)"
   CASE nType == 2
        cSmile := ";)"
ENDCASE
RETURN cSmile
```

Status

Ready

Compliance

__objDelMethod() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddInline()
__objAddMethod()
__objGetMethodList()
__objGetMsgList()
__objHasMethod()
__objModInline()
__objModMethod()

__objDelInline()

Delete a METHOD INLINE from class

Syntax

__objDelInline(<oObject>, <cSymbol>) --> oObject

Arguments

<oObject> is the object to work on.

<cSymbol> is the symbol name of METHOD or INLINE method to be deleted (removed) from the object.

Returns

__objDelInMethod() return a reference to <oObject>.

Description

__objDelInMethod() is a low level class support function that delete (remove) a METHOD or an INLINE method from an object. <oObject> is unchanged if a symbol with the name <cSymbol> does not exist in <oObject>.

Examples

```
// create a new THappy class and add a Smile method
oHappy := HBClass():New( "THappy" )
__objAddMethod( oHappy, "Smile", @MySmile() )
? __objHasMethod( oHappy, "Smile" )    // .T.
// remove Smile method
__objDelInMethod( oHappy, "Smile" )
? __objHasMethod( oHappy, "Smile" )    // .F.

STATIC FUNCTION MySmile( nType )
LOCAL cSmile
DO CASE
   CASE nType == 1
        cSmile := ":)"
   CASE nType == 2
        cSmile := ";)"
ENDCASE
RETURN cSmile
```

Status

Ready

Compliance

__objDelMethod() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddInline()
__objAddMethod()
__objGetMethodList()
__objGetMsgList()
__objHasMethod()
__objModInline()
__objModMethod()

__objDelData()

Delete a DATA (instance variable) from class

Syntax

__objDelMethod(<oObject>, <cDataName>) --> oObject

Arguments

<oObject> is the object to work on.

<cDataName> is the symbol name of DATA to be deleted (removed) from the object.

Returns

__objDelData() return a reference to <oObject>.

Description

__objDelData() is a low level class support function that delete (remove) a DATA from an object. <oObject> is unchanged if a symbol with the name <cDataName> does not exist in <oObject>.

Examples

```
// create a new THappy class and add a lHappy DATA
oHappy  := HBClass():New( "THappy" )
__objAddData( oHappy, "lHappy" )
? __objHasData( oHappy, "lHappy" )    // .T.
// remove lHappy DATA
__objDelData( oHappy, "lHappy" )
? __objHasData( oHappy, "lHappy" )    // .F.
```

Status

Ready

Compliance

__objDelData() is a Harbour extension.

Files

Library is rtl

See Also:

__objAddData()
__objGetMsgList()
__objGetValueList()
__objHasData()
__ObjSetValueList()

__objDerivedFrom()

Determine whether a class is derived from another class

Syntax

```
__objDerivedFrom( <oObject>, <xSuper> ) --> lIsParent
```

Arguments

<oObject> is the object to check.

<xSuper> is the object that may be a parent. can be either an Object or a Character string with the class name.

Returns

__objDerivedFrom() return a logical TRUE (.T.) if <oObject> is derived from <xSuper>.

Description

__objDerivedFrom() is a low level class support function that check is one class is a super class of the other, or in other words, does class <oObject> a child or descendant of <xSuper>.

Examples

```
// Create three classes and check their relations

#include "hbclass.ch"
FUNCTION main()
   local oSuper, oObject, oDress
   oSuper  := TMood():New()
   oObject := THappy():New()
   oDress  := TShirt():New()
   ? __objDerivedFrom( oObject, oSuper )    // .T.
   ? __objDerivedFrom( oSuper, oObject )    // .F.
   ? __objDerivedFrom( oObject, oDress )    // .F.
RETURN NIL

CLASS TMood
   METHOD New() INLINE Self
ENDCLASS

CLASS THappy FROM TMood
   METHOD Smile() INLINE qout( "*smile*" )
ENDCLASS

CLASS TShirt
   DATA Color
   DATA Size
   METHOD New() INLINE Self
ENDCLASS
```

Status

Ready

Compliance

__objDerivedFrom() is a Harbour extension.

Files

Library is rtl

See Also:

__objHasData()
__objHasMethod()

RDDLIST()

Return an array of the available Replaceable Database Drivers

Syntax

RDDLIST([<nRDDType>]) --> aRDDList

Arguments

<nRDDType> is an integer that represents the type of the RDD you wish to list. The constants RDT_FULL and RDT_TRANSFER represent the two types of RDDs currently available.

ÄÄ Constant Value Meaning ÄÄ RDT_FULL 1 Full RDD implementation RDT_TRANSFER 2 Import/Export only driver ÄÄ

RDT_FULL identifies full-featured RDDs that have all the capabilities associated with an RDD.

RDT_TRANSFER identifies RDDs of limited capability. They can only transfer records between files. You cannot use these limited RDD drivers to open a file in a work area. The SDF and DELIM drivers are examples of this type of RDD. They are only used in the implementation of APPEND FROM and COPY TO with SDF or DELIMITED files.

Returns

RDDLIST() returns a one-dimensional array of the RDD names registered with the application as <nRDDType>.

Description

RDDLIST() is an RDD function that returns a one-dimensional array that lists the available RDDs.

If you do not supply <nRDDType>, all available RDDs, regardless of type, are returned.

Examples

In this example RDDLIST() returns an array containing the character strings, "DBF", "SDF", "DELIM", "DBFCDX", and "DBFNTX":

```
REQUEST DBFCDX

.
. < statements >
.

aRDDs := RDDLIST()

      // Returns {"DBF", SDF", "DELIM", "DBFCDX", "DBFNTX" }
```

In this example, RDDLIST() returns an array containing the character strings, "SDF" and "DELIM":

```
#include "rddsys.ch"
.
. < statements >
.
aImpExp := RDDLIST( RDT TRANSFER )
```

Tests

Status

Ready

RDDNAME()

Return the name of the currently active RDD

Syntax

RDDNAME() --> cRDDName

Arguments

Returns

current or specified work area.

Description

RDDNAME() is an RDD function that returns a character string, cRDDName, the name of the active RDD in the current or specified work area.

You can specify a work area other than the currently active work area by aliasing the function.

Examples

```
USE Customer VIA "DBFNTX" NEW
USE Sales    VIA "DBFCDX" NEW

? RDDNAME()                          // Returns: DBFCDX
? Customer->( RDDNAME() )            // Returns: DBFNTX
? Sales->( RDDNAME() )               // Returns: DBFCDX
```

Tests

Status

Ready

See Also:

RDDLIST()

RDDSETDEFAULT()

Set or return the default RDD for the application

Syntax

```
RDDSETDEFAULT([<cNewDefaultRDD>])
--> cPreviousDefaultRDD
```

<cNewDefaultRDD> is a character string, the name of the RDD that is to be made the new default RDD in the application.

Returns

RDDSETDEFAULT() returns a character string, cPreviousDefaultRDD, the name of the previous default driver. The default driver is the driver that HARBOUR uses if you do not explicitly specify an RDD with the VIA clause of the USE command.

Description

RDDSETDEFAULT() is an RDD function that sets or returns the name of the previous default RDD driver and, optionally, sets the current driver to the new RDD driver specified by cNewDefaultRDD. If <cNewDefaultDriver> is not specified, the current default driver name is returned and continues to be the current default driver.

This function replaces the DBSETDRIVER() function.

Examples

```
// If the default driver is not DBFNTX, make it the default

IF ( RDDSETDEFAULT() != "DBFNTX" )
   cOldRdd := RDDSETDEFAULT( "DBFNTX" )
ENDIF
```

Tests

Status

Ready

See Also:

DBSETDRIVER()

__RDDSETDEFAULT()

Set or return the default RDD for the application

Syntax

```
__RDDSETDEFAULT([<cNewDefaultRDD>])
--> cPreviousDefaultRDD
```

<cNewDefaultRDD> is a character string, the name of the RDD that is to be made the new default RDD in the application.

Returns

__RDDSETDEFAULT() returns a character string, cPreviousDefaultRDD, the name of the previous default driver. The default driver is the driver that HARBOUR uses if you do not explicitly specify an RDD with the VIA clause of the USE command.

Description

RDDSETDEFAULT() is an RDD function that sets or returns the name of the previous default RDD driver and, optionally, sets the current driver to the new RDD driver specified by cNewDefaultRDD. If <cNewDefaultDriver> is not specified, the current default driver name is returned and continues to be the current default driver.

This function replaces the DBSETDRIVER() function.

Examples

```
// If the default driver is not DBFNTX, make it the default

IF ( __RDDSETDEFAULT() != "DBFNTX" )
     cOldRdd := __RDDSETDEFAULT( "DBFNTX" )
ENDIF
```

Tests

Status

Ready

See Also:

DBSETDRIVER()

DBEVAL()

Performs a code block operation on the current Database

Syntax

```
DBEVAL( <bBlock>,
[<bFor>], [<bWhile>],
[<nNext>], [<nRecord>],
[<lRest>] ) --> NIL
```

Arguments

<bBlock> Operation that is to be performed

<bFor> Code block for the For condition

<bWhile> Code block for the WHILE condition

<nNext> Number of NEXT records to process

<nRecord> Record number to work on exactly

<lRest> Toggle to rewind record pointer

Returns

DBEVAL() always returns NIL

Description

Performs a code block operation on the current Database

Examples

```
FUNCTION Main()
   LOCAL nCount

   USE Test

   dbGoto( 4 )
   ? RecNo()
   COUNT TO nCount
   ? RecNo(), nCount
   COUNT TO nCount NEXT 10
   ? RecNo(), nCount

   RETURN NIL
```

Status

Started

Compliance

DBEVAL is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

EVAL()

DBF()

Alias name of a work area

Syntax

Dbf() --> <cWorkArea>

Returns

<cWorkArea> Name of alias

Description

This function returns the same alias name ofthe currently selected work area.

Examples

```
FUNCTION Main()

   USE Test

   select 0
   qOut( IF(DBF()=="","No Name",DBF()))
   Test->(qOut(DBF())
   qOut(Alias(1))

   RETURN NIL
```

Status

Ready

Compliance

DBF() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

ALIAS()

DBAPPEND()

Appends a new record to a database file.

Syntax

DbAppend([<lLock>]) --> NIL

Arguments

<lLock> Toggle to release record locks

Returns

DbAppend() always returns NIL

Description

This function add a new record to the end of the database in the selected or aliased work area. All fields in that database will be given empty data values - character fields will be filled with blank spaces,date fields with CTOD('//'), numeric fields with 0,logical fields with .F., and memo fields with NULL bytes.The header of the database is not updated until the record is flushed from the buffer and the contents are written to the disk.

Under a networking enviroment, DBAPPEND() performs an additional operation: It attrmps to lock the newly added record. If the database file is currently locked or if a locking assignment if made to LASTREC()+1,NETERR() will return a logical true (.T.) immediately after the DBAPPEND() function. This function does not unlock the locked records.

If <lLock> is passed a logical true (.T.) value, it will release the record locks, which allows the application to main- tain multiple record locks during an appending operation. The default for this parameter is a logical false (.F.).

Examples

```
FUNCTION Main()

   USE Test
   local cName="HARBOUR",nId=10
   Test->(DbAppend())
   Replace Test->Name wit cName,Id with nId
   Use
   RETURN NIL
```

Status

Ready

Compliance

DBAPPEND() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

DBUNLOCK()
DBUNLOCKALL()

DBCLEARFILTER()

Clears the current filter condiction in a work area

Syntax

DbClearFilTer() --> NIL

Returns

DbClearFilTer() always returns NIL

Description

This function clears any active filter condiction for the current or selected work area.

Examples

```
Function Main()

 Use Test

 Set Filter to Left(Test->Name,2) == "An"

 Dbedit()

 Test->(DbClearFilter())

 USE

 Return Nil
```

Status

Ready

Compliance

DBCLEARFILTER() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

DBSETFILTER()
DBFILTER()

DBCLOSEALL()

Close all open files in all work areas.

Syntax

```
DbCloseAll() --> NIL
```

Returns

DBCLOSEALL() always return NIL

Description

This function close all open databases and all associated indexes.In addition, it closes all format files and moves the work area pointer to the first position

Examples

```
Function Main()

 Use Test New

 DbEdit()

 Use Test1 New

 DbEdit()

 DbCloseAll()

 USE

 Return Nil
```

Status

Ready

Compliance

DBCLOSEALL() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

DBUSEAREA()
DBCLOSEAREA()

DBCLOSEAREA()

Close a database file in a work area.

Syntax

DbCloseArea() --> NIL

Returns

DbCloseArea() always returns NIL.

Description

This function will close any database open in the selected or aliased work area.

Examples

```
Function Main()

 Use Test

 Dbedit()

 Test->(DbCloseArea())

 USE

 Return Nil
```

Status

Ready

Compliance

DBCLOSEAREA() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

DBUSEAREA()
DBCLOSEALL()

DBCOMMIT()

Updates all index and database buffers for a given workarea

Syntax

DBCOMMIT() --> NIL

Returns

DBCOMMIT() always returns NIL.

Description

This function updates all of the information for a give,selected, or active workarea.This operation includes all database and index buffers for that work area only. This function does not update all open work areas.

Examples

```
FUNCTION Main()
LOCAL cName:=SPACE(40)
LOCAL nId:=0
USE Test EXCLUSIVE NEW
//
@ 10, 10 GET cName
@ 11, 10 GET nId
READ
//
IF UPDATED()
   APPEND BLANK
   REPLACE Tests->Name WITH cName
   REPLACE Tests->Id WITH nId
   Tests->( DBCOMMIT() )
ENDIF
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

DBCLOSEALL()
DBCOMMITALL()
DBUNLOCK()

DBCOMMITALL()

Flushes the memory buffer and performs a hard-disk write

Syntax

DBCOMMIT() --> NIL

Returns

DBCOMMIT() always returns NIL.

Description

This function performs a hard-disk write for all work areas. Before the disk write is performed,all buffers are flushed. open work areas.

Examples

```
FUNCTION Main()
LOCAL cName:=SPACE(40)
LOCAL nId:=0
USE Test EXCLUSIVE NEW
USE TestId New INDEX Testid
//
@ 10, 10 GET cName
@ 11, 10 GET nId
READ
//
IF UPDATED()
   APPEND BLANK
   REPLACE Tests->Name WITH cName
   REPLACE Tests->Id WITH nId
   IF !TestId->(DBSEEK(nId))
      APPEND BLANK
      REPLACE Tests->Id WITH nId
   ENDIF
ENDIF
DBCOMMITALL()
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rdd

See Also:

DBCLOSEALL()
DBCOMMIT()
DBUNLOCK()

__DBCONTINUE()

Resume a pending LOCATE

Syntax

```
__DbCONTINUE() --> NIL
```

Returns

__DbCONTINUE() Always return NIL

Description

__DBCONTINUE is a database command that searches from the current record position for the next record meeting the most recent LOCATE condition executed in the current work area. It terminates when a match is found or end of file is encountered. If __DBCONTINUE is successful, the matching record becomes the current record and FOUND() returns true (.T.); if unsuccessful, FOUND() returns false (.F.).

Each work area may have an active LOCATE condition. In CA-Clipper, a LOCATE condition remains pending until a new LOCATE condition is specified. No other commands release the condition.

Notes

Scope and WHILE condition: Note that the scope and WHILE condition of the initial LOCATE are ignored; only the FOR condition is used with CONTINUE. If you are using a LOCATE with a WHILE condition and want to continue the search for a matching record, use SKIP and then repeat the original LOCATE statement adding REST as the scope.

This example scans records in Sales.dbf for a particular salesman and displays a running total sales amounts:

```
LOCAL nRunTotal := 0
USE Sales NEW
LOCATE FOR Sales->Salesman = "1002"
DO WHILE FOUND()
   ? Sales->Salesname, nRunTotal += Sales->Amount
   __DBCONTINUE()
ENDDO
```

This example demonstrates how to continue if the pending LOCATE scope contains a WHILE condition:

```
LOCAL nRunTotal := 0
USE Sales INDEX Salesman NEW
SEEK "1002"
LOCATE REST WHILE Sales->Salesman = "1002";
      FOR Sales->Amount > 5000
DO WHILE FOUND()
   ? Sales->Salesname, nRunTotal += Sales->Amount
   SKIP
   LOCATE REST WHILE Sales->Salesman = "1002";
      FOR Sales->Amount > 5000
ENDDO
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rdd

See Also:

EOF()
FOUND()

DBCREATE()

Creates an empty database from a array.

Syntax

DBCREATE(<cDatabase>, <aStruct>, [<cDriver>], [<lOpen>], [<cAlias>]) --> NIL

Arguments

<cDatabase> Name of database to be create

<aStruct> Name of a multidimensional array that contains the database structure

<cDriver> Name of the RDD

<lOpenNew> 3-way toggle to Open the file in New or Current workarea:

NIL	The file is not opened.
True	It is opened in a New area.
False	It is opened in the current area.

<cAlias> Name of database Alias

Returns

DBCREATE() always returns NIL.

Description

This function creates the database file specified as <cDatabase> from the multidimensional array <aStruct>.If no file extension is use with <cDatabase> the .DBF extension is assumed. The array specified in <aStruct> must follow a few guidelines when being built prior to a call to DBCREATE():

- All subscripts values in the second dimension must be set to proper values

- The fourth subscript value in the second dimension - which contains the decimal value-must he specified. even 1kw nonnumeric fields.

- The second subscript value in the second dimension-which contains the field data type-must contain a proper value: C, D, L, M or N It is possible to use additional letters (or clarity (e.g., 'Numeric' for 'N'): however, the first letter of this array element must be a proper value.

The DBCREATE( ) function does not use the decimal field to calculate the length of a character held longer than 256. Values up to the maximum length of a character field (which is 65,519 bytes) are stored directly in the database in the length attribute if that database was created via this function. However, a file containing fields longer than 256 bytes is not compatible with any interpreter.

The <cDriver> parameter specifies the name of the Replaceable Database Driver to use to create the database. If it is not specified, then the Replaceable Database Driver in the current work area is used.

The <lOpenNew> parameter specifies if the already created database is to be opened, and where. If NIL, the file is not opened. If True, it is opened in a New area, and if False it is opened in the current area (closing any file already occupying that area). The <cAlias> parameter specifies the alias name for the new opened database.

Examples

```
function main()

local nI, aStruct := { { "CHARACTER", "C", 25, 0 }, ;
                    { "NUMERIC",   "N",  8, 0 }, ;
                    { "DOUBLE",    "N",  8, 2 }, ;
                    { "DATE",      "D",  8, 0 }, ;
                    { "LOGICAL",   "L",  1, 0 }, ;
                    { "MEMO1",     "M", 10, 0 }, ;
                    { "MEMO2",     "M", 10, 0 } }
```

```
REQUEST DBFCDX

dbCreate( "testdbf", aStruct, "DBFCDX", .t., "MYALIAS" )

RETURN NIL
```

Status

Ready

Compliance

This function is Not CA-Clipper compliant

Files

Library is rdd Header is Dbstruct.ch

See Also:

AFIELDS()
DBSTRUCT()

DBDELETE()

Marks records for deletion in a database.

Syntax

DBDELETE() --> NIL

Returns

DBDELETE() always returns NIL.

Description

This function marks a record for deletion in the selected or aliased work area.If the DELETED setting is on, the record will still be visible until the record pointer in that work area is moved to another record.

In a networking situation, this function requires that the record be locked prior to issuing the DBDELETE() function.

Examples

```
nId:=10
USE TestId INDEX TestId NEW
IF TestId->(DBSEEK(nId))
   IF TestId->(RLOCK())
      DBDELETE()
   ENDIF
ENDIF
USE
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

DBRECALL()

DBFILTER()

Return the filter expression in a work area

Syntax

```
DBFILTER() --> cFilter
```

Returns

DBFILTER() returns the filter expression.

Description

This function return the expression of the SET FILTER TO command for the current or designated work area. If no filter condition is present,a NULL string will be returned.

Examples

```
USE Test INDEX Test NEW
SET FILTER TO Name= "Harbour"
USE TestId INDEX TestId NEW
SET FILTER TO Id = 1
SELECT Test
//
? DBFILTER()
? TestId->(DBFILTER())
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

ARRAY()
ARRAY()

DBGOBOTTOM()

Moves the record pointer to the bottom of the database.

Syntax

```
DBGOBOTTOM() --> NIL
```

Returns

DBGOBOTTOM() always returns NIL.

Description

This function moves the record pointer in the selected or aliased work area to the end of the file.The position of the record pointer is affected by the values in the index key or by an active FILTER condition.Otherwise,if no index is active or if no filter condition is present,the value of the record pointer will be LASTREC().

Examples

```
USE Tests
DBGOTOP()
? RECNO()
DBGOBOTTOM()
? RECNO()
USE
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

- BOF()
- EOF()
- DBSKIP()
- DBSEEK()
- DBGOTOP()

DBGOTO()

Position the record pointer to a specific location.

Syntax

DBGOTO(<xRecordNumber>) --> NIL

Arguments

<xRecordNumber> Record number or unique identity

Returns

DBGOTO() always returns NIL.

Description

This function places the record pointer,if working with a .DBF file, in selected or aliased work area at the record number specified by <xRecordNumber>.The position if not affected by an active index or by any enviromental SET condicition.

Issuing a DBGOTO(RECNO()) call in a network enviroment will refresh the database and index buffers.This is the same as a DBSKIP(0) call. The parameter <xRecordNumber> may be something other than a record number.In some data formats, for example, the value of <xRecordNumber> is a unique primary key while in other formats,<xRecordNumber> could be an array offset if the data set was an array.

Examples

```
The following example uses DBGOTO() to iteratively process
every fourth record:

DBUSEAREA( .T., "DBFNTX", "Sales", "Sales", .T. )
//
// toggle every fourth record
DO WHILE !EOF()
   DBGOTO( RECNO() + 4 )
   Sales->Group := "Bear"
ENDDO
```

Status

Ready
This function is CA-Clipper compliant.

Files

Library is rdd

See Also:

- BOF()
- EOF()
- DBGOTOP()
- DBGOBOTTOM()
- DBSEEK()
- DBSKIP()

DBGOTOP()

Moves the record pointer to the bottom of the database.

Syntax

DBGOTOP() --> NIL

Returns

DBGOTOP() always returns NIL.

Description

This function moves the record pointer in the selected or aliased work area to the top of the file.The position of the record pointer is affected by the values in the index key or by an active FILTER condition.Otherwise,if no index is active or if no filter condition is present,the value of RECNO() will be 1.

Examples

```
USE Tests
DBGOTOP()
? RECNO()
DBGOBOTTOM()
? RECNO()
USE
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

- BOF()
- EOF()
- DBSKIP()
- DBSEEK()
- DBGOBOTTOM()

DBRECALL()

Recalls a record previousy marked for deletion.

Syntax

DBRECALL() --> NIL

Returns

DBRECALL() always returns NIL.

Description

This function unmarks those records marked for deletion and reactivates them in the aliased or selected work area. If a record is DELETED and the DELETED setting is on, the record will still be visible for a DBRECALL() provided that the database record pointer has not been skipped. Once a record marked for deletion with the DELETE setting ON has been skipped, it no longer can be brought back with DBRECALL().

Examples

```
USE Test NEW
DBGOTO(10)
DBDELETE()
? DELETED()
DBRECALL()
? DELETED()
USE
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

DBDELETE()

DBRLOCK()

This function locks the record basedon identify

Syntax

DBRLOCK([<xIdentity>]) --> lSuccess

Arguments

<xIdentity>	Record indetifier

Returns

DBRLOCK() returns a logical true (.T.) if lock was successful

Description

This function attempts to lock a record which is indentified by <xIdentity> in the active data set.If the lock is successful the function will return a logical true (.T.) value;otherwise a logical false (.F.) will be returned.If <xIdentity> is not passed it will be assumed to lock the current active record/data item.

Examples

```
FUNCTION Main()
LOCAL x:=0
USE Tests New
FOR x:=1 to reccount()
  IF !DBRLOCK()
    DBUNLOCK()
  ENDIF
NEXT
USE
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

DBUNLOCK()
DBUNLOCKALL()
FLOCK()
RLOCK()

DBRLOCKLIST()

This function return a list of records in the database work area

Syntax

DBRLOCKLIST() --> aRecordLocks

Returns

<aRecordList> is an array of lock records

Description

This function will return an array of locked records in a given and active work area.If the return array is an empty array (meaning no elements in it),then there are no locked record in that work area.

Examples

```
FUNCTION Main()
LOCAL aList:={}
LOCAL x:=0
USE Tests NEW
DBGOTO(10)
RLOCK()
DBGOTO(100)
RLOCK()
aList:=DBRLOCKLIST()
FOR x:=1 TO LEN(aList)
   ? aList[x]
NEXT
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

RLOCK()
DBRLOCK()
DBRUNLOCK()

DBRUNLOCK()

Unlocks a record base on its indentifier

Syntax

DBRUNLOCK([<xIdentity>]) --> NIL

Arguments

<xIdentity> Record indentifier,tipicaly a record number

Returns

DBRUNLOCK() always returns NIL.

Description

This function will attempt to unlock the record specified as <xIdentity>,which in a .DBF format is the record number.If not specified,them the current active record/data item will be unlocked

Examples

```
FUNCTION Main()
USE Tests New
DBGOTO(10)
IF RLOCK()
   ? Tests->ID
   DBRUNLOCK()
ENDIF
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

RLOCK()
DBRLOCK()
DBRLOCKLIST()

DBSEEK()

Searches for a value based on an active index.

Syntax

DBSEEK(<expKey>, [<lSoftSeek>],[<lFindLast>]) --> lFound

Arguments

<expKey> Any expression

<lSoftSeek> Toggle SOFTSEEK condition

<lFindLast> is an optional logical value that set the current record position to the last record if successful

Returns

DBSEEK() returns logical true (.T.) if found, otherwise false

Description

This function searches for the first record in a database file whose index key matches <expKey>. If the item is found, the function will return a logical true (.T.), the value of FOUND() wilI be a logical true (.T.), and the value of EOF() wilI be a logical false (.F.). If no item is found. then the function will return a logical false, the value of FOUND( ) will be a logical false (.F.), and the value of EOF( ) will be a logical true (.T.).

This function always "rewinds" the database pointer and starts the search from the top of the file.

If the SOFTSEEK flag is on or if <lSoftSeek> is set to a logical true (.T.) the value of FOUND() wilI be a logical false and EOF() will be false if there is an item in the index key with a greater value than the key expression <expKey>; at this point the record pointer will position itself on that record. However, if there is no greater key in the index,EOF() will return a logical true (.T.) value. If <lSoftSeek> is not passed, the function will look to the internal status of SOFTSEEK before performing the operation. The default of <lSoftSeek> is a logical false (.F.)

Examples

```
FUNCTION Main()
USE Tests New INDEX Tests
DBGOTO(10)
nId:=Tests->nId
IF Tests->(DBSEEK(nId))
  IF RLOCK()
     ? Tests->Name
     DBRUNLOCK()
  ENDIF
ENDIF
USE
RETURN NIL

ACCEPT "Employee name: " TO cName
IF ( Employee->(DBSEEK(cName)) )
   Employee->(ViewRecord())
ELSE
   ? "Not found"
END
```

Status

Started

Compliance

DBSEEK() is Compatible with CA-Clipper 5.3

Files

Library is rdd

See Also:

DBGOBOTTOM()
DBGOTOP()
DBSKIP()
EOF()
BOF()
FOUND()

DBSELECTAREA()

Change to another work area

Syntax

DBSELECTAREA(<xArea>) --> NIL

Arguments

<xArea> Alias or work area

Returns

DBSELECTAREA() always returns NIL.

Description

This function moves the Harbour internal primary focus to the work area designated by <xArea>. If <xArea> is numeric, them it will select the numeric work area;if <xArea> is character,then it will select the work area with the alias name.

DBSELECTAREA(0) will select the next avaliable and unused work area. Up to 255 work areas are supported.Each work area has its own alias and record pointer, as well as its own FOUND(), DBFILTER(), DBRSELECT() and DBRELATION() function values.

Examples

```
FUNCTION Main()
LOCAL nId
USE Tests NEW INDEX Tests
USE Tests1 NEW INDEX Tests1
DBSELECTAREA(1)
nId:=Tests->Id
DBSELECTAREA(2)
IF DBSEEK(nId)
   ? Tests1->cName
ENDIF
DBCLOSEALL()
RETURN NIL
```

Status

Ready

Compliance

This function is CA-CLIPPER compatible.

Files

Library is rdd

See Also:

DBUSEAREA()
SELECT()

DBSETDRIVER()

Establishes the RDD name for the selected work area

Syntax

```
DBSETDRIVER( [<cDriver>] ) --> cCurrentDriver
```

Arguments

<cDriver> Optional database driver name

Returns

DBSETDRIVER() returns the name of active driver

Description

This function returns the name of the current database driver for the selected work area. The default will be "DBFNTX". If specified, <cDriver> contains the name of the database driver that should be used to activate and manage the work area.If the specified driver is not avaliable,this function will have no effect.

Examples

```
DBSETDRIVER("ADS")
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Files

Library is rdd

See Also:

DBUSEAREA()

DBSKIP()

Moves the record pointer in the selected work area.

Syntax

DBSKIP([<nRecords>]) --> NIL

Arguments

<nRecords> Numbers of records to move record pointer.

Returns

DBSKIP() always returns NIL.

Description

This function moves the record pointer <nRecords> in the selected or aliased work area.The default value for <nRecords> will be 1. A DBSKIP(0) will flush and refresh the internal database bufer and make any changes made to the record visible without moving the record pointer in either direction.

Examples

```
FUNCTION Main()
USE Tests NEW
DBGOTOP()
WHILE !EOF()
  ? Tests->Id, Tests->Name
  DBSKIP()
ENDDO
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-CLIPPER compatible

Files

Library is rdd

See Also:

BOF()
DBGOBOTTOM()
DBGOTOP()
DBSEEK()
EOF()

DBSETFILTER()

Establishes a filter condition for a work area.

Syntax

DBSETFILTER(<bCondition>, [<cCondition>]) --> NIL

Arguments

<bCondition> Code block expression for filtered evaluation.

<cCondition> Optional character expression of code block.

Returns

DBSETFILTER() always returns NIL.

Description

This function masks a database so that only those records that meet the condition prescribed by the expression in the code block <bCondition> and literally expressed as <cCondition> are visible. If <cCondition> is not passed to this function,then the DBFILTER() function will return an empty string showing no filter in that work area which in fact,would be not correct.

Examples

```
FUNCTION Main()
USE Tests NEW
DBSETFILTER( {|| Tests->Id <100 }, "Tests->Id <100" )
DBGOTOP()
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rdd

See Also:

DBFILTER()
DBCLEARFILTER()

DBSTRUCT()

Creates a multidimensional array of a database structure.

Syntax

DBSTRUCT() --> aStruct

Returns

DBSTRUCT() returns an array pointer to database structure

Description

This function returns a multidimensional array.This array has array pointers to other arrays,each of which contains the characteristic of a field in the active work area.The lenght of this array is based in the number of fields in that particular work area.In other words, LEN(DBSTRUCT()) is equal to the value obtained from FCOUNT(). Each subscript position

Examples

```
FUNCTION Main()
LOCAL aStru,x
USE Tests NEW
aStru:=DBSTRUCT()
FOR x:=1 TO LEN(aStru)
  ? aStru[x,1]
NEXT
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd Header is DbStruct.ch

See Also:

AFIELDS()

DBUNLOCK()

Unlock a record or release a file lock

Syntax

DBUNLOCK() --> NIL

Returns

DBUNLOCK() always returns NIL.

Description

This function releases the file or record lock in the currently selected or aliased work area.It will not unlock an associated lock in a related databases.

Examples

```
nId:=10
USE TestId INDEX TestId NEW
IF TestId->(DBSEEK(nId))
   IF TestId->(RLOCK())
      DBDELETE()
   ELSE
       DBUNLOCK()
   ENDIF
ENDIF
USE
```

Status

Ready

Compliance

This function is CA-Clipper compatible.

Files

Library is rdd

See Also:

DBUNLOCKALL()
FLOCK()
RLOCK()

DBUNLOCKALL()

Unlocks all records and releases all file locks in all work areas.

Syntax

DBUNLOCKALL() --> NIL

Returns

DBUNLOCKALL() always returns NIL.

Description

This function will remove all file and record locks in all work area.

Examples

```
nId:=10
USE Tests INDEX TestId NEW
USE Tests1 INDEX Tests NEW
IF TestId->(DBSEEK(nId))
   IF TestId->(RLOCK())
      DBDELETE()
   ELSE
      DBUNLOCK()
   ENDIF
ELSE
   DBUNLOCKALL()
ENDIF
USE
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is rdd

See Also:

DBUNLOCK()
FLOCK()
RLOCK()

DBUSEAREA()

Opens a work area and uses a database file.

Syntax

DBUSEAREA([<lNewArea>], [<cDriver>], <cName>, [<xcAlias>], [<lShared>], [<lReadonly>]) --> NIL

Arguments

<lNewArea>	A optional logical expression for the new work area
<cDriver>	Database driver name
<cName>	File Name
<xcAlias>	Alias name
<lShared>	Shared/exclusive status flag
<lReadonly>	Read-write status flag.

Returns

DBUSEAREA() always returns NIL.

Description

This function opens an existing database named <cName> in the current work area. If <lNewArea> is set to a logical true (.T.) value, then the database <cName> will be opened in the next available and unused work area. The default value of <lNewArea> is a logical false (.F.). If used, <cDriver> is the name of the database driver associated with the file <cName> that is opened. The default for this will be the value of DBSETDRlVER().

IF used, <xcAlias> contains the alias name for that work area, If not specified, the root name of the database specified in <cName> will be used.

If <lShared> is set to a logical true (.T.) value, the database that is specified in <cName> will be opened by the user EXCLUSIVELY. Thus locking it from all other nodes or users on the network. If <lShared> is set to a logical false (.F.) value, then the database will be in SHARED mode. If <lShared> is not passed, then the function will turn to the internal setting of SET EXCLUSIVE to determine a setting.

If <lReadOnly> is specified, the file will be set to READ ONLY mode. If it is not specified, the file will he opened in normal read-write mode.

Examples

```
DBUSEAREA(.T.,,"Tests")
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

- DBCLOSEAREA()
- DBSETDRIVER()
- SELECT()
- SET()

__DBZAP()

Remove all records from the current database file

Syntax

__DbZap() --> NIL

Returns

__DbZap() will always return NIL

Description

__DbZap() is a database command that permanently removes all records from files open in the current work area. This includes the current database file, index files, and associated memo file. Disk space previously occupied by the ZAPped files is released to the operating system.

__DbZap() performs the same operation as DELETE ALL followed by PACK but is almost instantaneous.

To ZAP in a network environment, the current database file must be USEd EXCLUSIVEly.

This example demonstrates a typical ZAP operation in a network environment:

```
USE Sales EXCLUSIVE NEW
IF !NETERR()
   SET INDEX TO Sales, Branch, Salesman
   __dbZAP()
   CLOSE Sales
ELSE
   ? "Zap operation failed"
   BREAK
ENDIF
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is rdd

AFIELDS()

Fills referenced arrays with database field information

Syntax

AFields(<aNames>[,<aTypes>][,<aLen>][,<aDecs>]) --> <nFields>

Arguments

<aNames> Array of field names

<aTypes> Array of field names

<aLens> Array of field names

<aDecs> Array of field names

Returns

<nFields> Number od fields in a database or work area

Description

This function will fill a series of arrays with field names,field types,field lenghts, and number of field decimal positions for the currently selected or designed database. Each array parallels the different descriptors of a file's structure.The first array will consist of the names of the fields in the current work area.All other arrays are optional and will be filled with the corrensponding data. This function will return zero if no parameters are specified or if no database is avaliable in the current work area.Otherwise, the number of fields or the lenght of the shortest array argument, witchever is smaller, will be returned.

Examples

```
FUNCTION Main()
   LOCAL aNames:={},aTypes:={},aLens:={},aDecs:={},nFields:=0

   USE Test

   dbGoTop()
   nFields:=aFields(aNames,aTypes,aLens,aDecs)

   ? "Number of fields", nFields

   RETURN NIL
```

Status

Ready

Compliance

AFIELDS() is fully CA-Clipper compliant.

Files

Library is rdd

ALIAS()

Returns the alias name of a work area

Syntax

```
Alias([<nWorkArea>]) --> <cWorkArea>
```

Arguments

<nWorkArea> Number of a work area

Returns

<cWorkArea> Name of alias

Description

This function returns the alias of the work area indicated by <nWorkArea> If <nWorkArea> is not provided, the alias of the current work area is returned.

Examples

```
FUNCTION Main()

USE Test
select 0
qOut( IF(Alias()=="","No Name",Alias()))
Test->(qOut(Alias())
qOut(Alias(1))

RETURN NIL
```

Status

Ready

Compliance

ALIAS() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

DBF()

BOF()

Test for the beggining-of-file condition

Syntax

BOF() --> <lBegin>

Returns

BOF() Logical true (.T.) or false (.F.)

Description

This function determines if the beggining of the file marker has been reached. If so, the function will return a logical true (.T.); otherwise, a logical false(.F.) will be returned. By default, BOF() will apply to the currently selected database unless the function is preceded by an alias

Examples

```
 FUNCTION Main()
   USE Tests NEW
   DBGOTOP()
   ? "Is Eof()",EOF()
   DBGOBOTTOM()
   ? "Is Eof()",EOF()
   USE
RETURN NIL
```

Status

Ready

Compliance

BOF() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

EOF()
FOUND()
LASTREC()

ZAP

Remove all records from the current database file

Syntax

ZAP

Description

This command removes all of the records from the database in the current work area.This operation also updates any index file in use at the time of this operation.In addition, this command removes all items within an associated memo file. In a network enviroment,any file that is about to be ZAPped must be used exclusively.

Examples

```
USE Tests NEW index Tests
ZAP
USE
```

Status

Ready

Compliance

This command is CA Clipper compliant

See Also:

ARRAY()
PACK
ARRAY()

DELETED()

Tests the record's deletion flag.

Syntax

DELETED() --> lDeleted

Returns

DELETED() return a logical true (.T.) or false (.F.).

Description

This function returns a logical true (.T.) is the current record in the selected or designated work area ha ben marked for deletion.If not, the function will return a logical false (.F.).

Examples

```
FUNCTION Main()
USE Test New
DBGOTO()
DBDELETE()
? "Is Record Deleted",Test->(DELETED())
DBRECALL()
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

DBDELETE()

EOF()

Test for end-of-file condition.

Syntax

EOF() --> <lEnd>

Returns

<lEnd> A logical true (.T.) or false (.F.)

Description

This function determines if the end-of-file marker has been reached. If it has, the function will return a logical true (.T.); otherwise a logical false (.F.) will be returnd

Examples

```
  FUNCTION Main()
    USE Tests NEW
    DBGOTOP()
    ? "Is Eof()",EOF()
    DBGOBOTTOM()
    ? "Is Eof()",EOF()
    USE
RETURN NIL
```

Status

Ready

Compliance

EOF() is fully CA-Clipper compliant.

Files

Library is rdd

See Also:

BOF()
FOUND()
LASTREC()

FCOUNT()

Counts the number of fields in an active database.

Syntax

```
FCOUNT() --> nFields
```

Returns

<nFields> Return the number of fields

Description

This function returns the number of fields in the current or designated work area.If no database is open in this work area, the function will return 0.

Examples

```
FUNCTION Main()
  USE Tests NEW
  ? "This database have ",Tests->(FCOUNT()),"Fields"
  USE
RETURN Nil
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

FIELDNAME()
TYPE()

FIELDGET()

Obtains the value of a specified field

Syntax

FIELDGET(<nField>) --> ValueField

Arguments

<nField> Is the numeric field position

Returns

<ValueField>   Any expression

Description

This function returns the value of the field at the <nField>th location in the selected or designed work area.If the value in <nField> does not correspond to n avaliable field position in this work area, the function will return a NIL data type.

Examples

```
FUNCTION Main()
USE Test NEW
? Test->(FieldGet(1))
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper Compliant.

Files

Library is rdd

See Also:

FIELDPUT()

FIELDNAME()

Return the name of a field at a numeric field location.

Syntax

FIELDNAME/FIELD(<nPosition>) --> cFieldName

Arguments

<nPosition> Field order in the database.

Returns

<cFieldName> returns the field name.

Description

This function return the name of the field at the <nPosition>th position. If the numeric value passed to this function does not correspond to an existing field in the designated or selected work area,this function will return a NULL byte.

Examples

```
FUNCTION Main()
  LOCAL x
  USE Tests NEW
  FOR x := 1 to Tests->(FCOUNT())
    ? "Field Name",FieldName(x)
  NEXT
  USE
RETURN Nil
```

Status

Ready

Compliance

This function is CA-Clipper compatible.

Files

Library is rdd

See Also:

DBSTRUCT()
FCOUNT()
LEN()
VALTYPE()

FIELDPOS()

Return the ordinal position of a field.

Syntax

FIELDPOS(<cFieldName>) --> nFieldPos

Arguments

<cFieldName> Name of a field.

Returns

<nFieldPos> is ordinal position of the field.

Description

This function return the ordinal position of the specified field <cField> in the current or aliased work areaIf there isn't field under the name of <cField> or of no database is open in the selected work area, the func- tion will return a 0.

Examples

```
FUNCTION Main()
USE Test NEW
? Test->(FIELDPOS("ID"))
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-Clipper compliant.

Files

Library is rdd

See Also:

FIELDGET()
FIELDPUT()

FIELDPUT()

Set the value of a field variable

Syntax

FIELDPUT(<nField>, <expAssign>) --> ValueAssigned

Arguments

<nField> The field numeric position

<expAssign> Expression to be assigned to the specified field

Returns

<ValueAssigned> Any expression

Description

This function assings the value in <expAssing> to the <nField>th field in the current or designated work area.If the operation is successful,the return value of the function will be the same value assigned to the specified field.If the operation is not successful, the function will return a NIL data type

Examples

```
USE Tests New
FIELDPUT(1,"Mr. Jones")
USE
```

Status

Ready

Compliance

This function is CA-Clipper compatible.

Files

Library is rdd

See Also:

FIELDGET()

FLOCK()

Locks a file

Syntax

FLOCK() --> lSuccess

Returns

<lSuccess> A true (.T.) value, if the lock was successful;otherwise false (.F.)

Description

This function returns a logical true (.T.0 if a file lock is attempted and is successfully placed on the current or designated database.This function will also unlock all records locks placed by the same network station.

Examples

```
USE Tests New
IF FLOCK()
   SUM Tests->Ammount
ENDIF
USE
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Files

Library is rdd

See Also:

RLOCK()

FOUND()

Determine the success of a previous search operation.

Syntax

FOUND() --> lSuccess

Arguments

Returns

<lSuccess> A logical true (.T.) is successful;otherwise, false (.F.)

Description

This function is used to test if the previous SEEK,LOCATE,CONTINUE, or FIND operation was successful.Each wrk area has its own FOUND() flag,so that a FOUND() condition may be tested in unselected work areas by using an alias.

Examples

```
nId:=100
USE Tests NEW INDEX Tests
SEEK nId
IF FOUND()
  ? Tests->Name
ENDIF
USE
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Files

Library is rdd

See Also:

EOF()

HEADER()

Return the length of a database file header

Syntax

```
HEADER() --> nBytes
```

Returns

<nBytes> The numeric size of a database file header in bytes

Description

This function returns the number of bytes in the header of the selected database ot the database in the designated work area.

If used in conjunction with the LASTREC(),RECSIZE() and DISKSPACE() functions,this functions is capable of implementing a backup and restore routine.

Examples

```
USE Tests New
? Header()
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Files

Library is rdd

See Also:

DISKSPACE()
LASTREC()
RECSIZE()

LASTREC()

Returns the number of records in an active work area or database.

Syntax

LASTREC() | RECCOUNT()* --> nRecords

Returns

<nRecords > The number of records

Description

This function returns the number of records present in the database in the selected or designated work area.If no records are present the value of this function will be 0.Additionaly,if no database is in use in the selected or designated work area,this function will return a 0 value as well.

Examples

```
USE Tests NEW
? LASTREC(), RECCOUNT()
```

Status

Ready

Compliance

This function is CA Clipper compatible

Platforms

All

Files

Library is rdd

See Also:

EOF()

LUPDATE()

Yields the date the database was last updated.

Syntax

```
LUPDATE() --> dModification
```

Arguments

Returns

<dModification> The date of the last modification.

Description

This function returns the date recorded by the OS when the selected or designated database was last written to disk.This function will only work for those database files in USE.

Examples

```
Function Main

Use Tests New
? Lupdate() // 04/25/2000
Use
Return Nil
```

Status

Ready

Compliance

This function is CA Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

FIELDNAME()
LASTREC()
RECSIZE()

NETERR()

Tests the success of a network function

Syntax

NETERR([<lNewError>]) --> lError

Arguments

<lNewError> Is a logical Expression.

Returns

<lError> A value based on the success of a network operation or function.

Description

This function return a logical true (.T.) is a USE,APPEND BLANK, or a USE...EXCLUSIVE command is issue and fails in a network enviroment. In the case of USE and USE...EXCLUSIVE commands,a NETERR() value of .T. would be returned if another node of the network has the exclusive use of a file.And the case of the APPEND BLANK command, NETERR() will return a logical true (.T.) if the file or record is locked by another node or the value of LASTREC() has been advanced The value of NETERR() may be changed via the value of <lNewError>. This allow the run-time error-handling system to control the way certains errors are handled.

Examples

```
USE TEST NEW Index Test
If !NetErr()
    Seek Test->Name="HARBOUR"
    If Found()
       ? Test->Name
    Endif
Endif
USE
```

Status

Ready

Compliance

This function is CA Clipper compliant

Files

Library is rdd

See Also:

FLOCK()
RLOCK()

RECCOUNT()

Counts the number of records in a database.

Syntax

RECCOUNT()* | LASTREC() --> nRecords

Arguments

Returns

<nRecords> The number of records

Description

This function returns the number of records present in the database in the selected or designated work area.If no records are present the value of this function will be 0.Additionaly,if no database is in use in the selected or designated work area,this function will return a 0 value as well.

Examples

```
Use Test NEW
USE Harbour NEW
? Reccount()
? Test->(RECCOUNT())
CLOSE ALL
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Files

Library is rdd

See Also:

EOF()
LASTREC()
RECNO()
DBGOBOTTOM()

RECNO()

Returns the current record number or identity.

Syntax

RECNO() --> Identity

Arguments

Returns

RECNO() The record number or indentity

Description

This function returns the position of the record pointer in the currently selected ot designated work area. If the database file is empty and if the RDD is the traditional .DBF file,the value of this function will be 1.

Examples

```
USE Tests NEW
DBGOTOP()
RECNO()            // Returns 1
DBGOTO(50)
RECNO()            // Returns 50
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Files

Library is rdd

See Also:

DBGOTO()
DBGOTOP()
DBGOBOTTOM()
LASTREC()
EOF()
BOF()

RECSIZE()

Returns the size of a single record in an active database.

Syntax

RECSIZE() --> nBytes

Arguments

Returns

<nBytes> The record size.

Description

This function returns the number os bytes used by a single record in the currently selected or designated database file.If no database is in use in this work area,the return value from this function will be 0.

Examples

```
USE Tests NEW
DBGOTOP()
RECSIZE()            // Returns 1
DBGOTO(50)
RECSIZE()
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Files

Library is rdd

See Also:

DISKSPACE()
FIELDNAME()
HEADER()
LASTREC()

RLOCK()

Lock a record in a work area

Syntax

```
RLOCK() --> lSuccess
```

Arguments

Returns

RLOCK() True (.T.) if record lock is successful; otherwise, it returns false (.F.).

Description

This function returns a logical true (.T.) if an attempt to lock a specific record in a selected or designated work area is successful. It will yield a false (.F.) if either the file or the desired record is currently locked. A record that is locked remains locked until another RLOCK() is issued or until an UNLOCK command is executed. On a Network enviroment the follow command need that the record is locked:

@...GET

DELETE (single record)

RECALL (single record)

REPLACE (single record)

Examples

```
nId:=10
USE TestId INDEX TestId NEW
IF TestId->(DBSEEK(nId))
   IF TestId->(RLOCK())
      DBDELETE()
   ENDIF
ENDIF
USE
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Files

Library is rdd

See Also:

FLOCK()

SELECT()

Returns the work area number for a specified alias.

Syntax

SELECT([<cAlias>]) --> nWorkArea

Arguments

<cAlias> is the target work area alias name.

Returns

SELECT() returns the work area number.

Description

This function returns the work area number for the specified alias name <cAlias>.If no parameter is specified,the current work area will be the return value of the function.

Examples

```
USE TESTS NEW
USE NAMES NEW
cOldArea:=SELECT("NAMES")
select TEST
LIST
SELECT cOldArea
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Files

Library is rdd

See Also:

ALIAS()
USED()

USED()

Checks whether a database is in use in a work area

Syntax

USED() --> lDbfOpen

Arguments

Returns

<lDbfOpen> True is a database is Used;otherwise False

Description

This function returns a logical true (.T.) if a database file is in USE in the current or designated work area. If no alias is specified along with this function , it will default to the currently selected work area.

Examples

```
Use TESTS NEW
USE Names New
? USED()    // .T.
? TESTS->(USED()) //.t.
CLOSE
? USED()  // .F.
Select TESTS
? USED() //.T.
```

Status

Ready

Compliance

This function is Ca-clipper Compliant

Files

Library is rdd

See Also:

ALIAS()
SELECT()

PACK

Remove records marked for deletion from a database

Syntax

```
PACK
```

Description

This command removes records that were marked for deletion from the currently selected database.This command does not pack the contents of a memo field;those files must be packed via low-level fuctions.

All open index files will be automatically reindexed once PACK command has completed its operation.On completion,the record pointer is placed on the first record in the database.

Examples

```
USE Tests NEW index Tests
DBGOTO(10)
DELETE NEXT 10
PACK
USE
```

Status

Ready

Compliance

This command is CA Clipper compliant

See Also:

DBEVAL()
ARRAY()
DELETED()
ZAP
ARRAY()

ORDBAGEXT()

Returns the Order Bag extension

Syntax

ORDBAGEXT() --> cBagExt

Arguments

Returns

<cBagExt> The Rdd extension name.

Description

This function return th character name of the RDD extension for the order bag.This is determined by the active RDD for the selected work area.

This function replaces the Indexord() function.

Examples

```
USE Tests NEW VIA "DBFNTX"
? ORDBAGEXT()      //  Returns .ntx
DBCLOSEAREA()
USE Tests NEW VIA "DBFCDX"
? ORDBAGEXT()      //  Returns .cdx
DBCLOSEAREA()
```

Status

Started

Compliance

This function is CA Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

INDEXEXT()
ORDBAGNAME()

ORDBAGNAME()

Returns the Order Bag Name.

Syntax

ORDBAGNAME(<nOrder> | <cOrderName>) --> cOrderBagName

Arguments

<nOrder> A numeric value representing the Order bag number.

<cOrderName> The character name of the Order Bag.

Returns

ORDBAGNAME() returns the Order bag name

Description

This function returns the name of the order bag for the specified work area. If <nOrder> is specidied,it will represent the position in the order list of the target order.If <cOrderName> is specified, it will represent the name of the target order.In essence,it will tell the name of the database (if That Rdd is in use) for a given index name or index order number.If <cOrderName> is not specified or <nOrder> is 0, the Current active order will be used.

Examples

```
USE Tests VIA "DBFCDX" NEW
Set index to TESTs
ORDBAGNAME( "TeName" )        // Returns: Customer
ORDBAGNAME( "TeLast" )        // Returns: Customer
ORDBAGNAME( "teZip" )         // Returns: Customer
Set Order to Tag TeName
? OrderBagName() //Return Custumer
```

Tests

See Examples

Status

Started

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

INDEXORD()
ORDBAGEXT()
ALIAS()

ORDCONDSET()

Set the Condition and scope for an order

Syntax

```
ORDCONSET([<cForCondition>],
[<bForCondition>],
[<lAll>],
[<bWhileCondition>],
[<bEval>],
[<nInterval>],
[<nStart>],
[<nNext>],
[<nRecord>],
[<lRest>],
[<lDescend>],
[<lAdditive>],
[<lCurrent>],
[<lCustom>],
[<lNoOptimize>])
```

Arguments

<cForCondition> is a string that specifies the FOR condition for the order. <bForCondition> is a code block that defines a FOR condition that each record within the scope must meet in order to be processed. If a record does not meet the specified condition,it is ignored and the next record is processed.Duplicate keys values are not added to the index file when a FOR condition is Used.

Returns

Description

Status

Started
ORDCONDSET() is CA-Clipper compliant

Files

Library is rdd

ORDCREATE()

Create an Order in an Order Bag

Syntax

ORDCREATE(<cOrderBagName>,[<cOrderName>], <cExpKey>, [<bExpKey>], [<lUnique>]) --> NIL

Arguments

<cOrderBagName> Name of the file that contains one or more Orders.

<cOrderName> Name of the order to be created.

<cExpKey> Key value for order for each record in the current work area

<bExpKey> Code block that evaluates to a key for the order for each record in the work area.

<lUnique> Toggle the unique status of the index.

Returns

ORDCREATE() always returns NIL.

Description

This function creates an order for the current work area.It is similar to the DBCREATEINDEX() except that this function allows different orders based on the RDD in effect.The name of the file <cOrderBagName> or the name of the order <cOrderName> are technically both considered to be "optional" except that at least one of two must exist in order to create the order.

The parameter <cExpKey> is the index key expression;typically in a .DBF driver,the maximum length of the key is 255 characters.

If <bExpKey> is not specified,then the code block is create by macro expanding the value of <cExpKey>.

If <lUnique> is not specified,then the current internal setting of SET UNIQUE ON or OFF will be observed.

The active RDD driver determines the capacity in the order for a specific order bag.

If the name <cOrderBagName> is found in the order bag can contain a single order,the the name <cOrderBagName> is erased and a new order is added to the order list in the current or specified work area.On the other hand,if it can contain multiples tags and if <cOrderBagName> does not already exist in the order list,then it is added.It is does exist,then the <cOrderBagName> replaces the former name in the order list in the current or specified work area.

Examples

```
USE TESTS VIA "DBFNDX" NEW
ORDCREATE( "FNAME",, "Tests->fName" )

USE TEsts VIA "DBFCDX" NEW
ORDCREATE( , "lName", "tests->lName" )
```

Tests

See examples

Status

Started

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

ARRAY()
ORDNAME()
ORDSETFOCUS()

ORDDESTROY()

Remove an Order from an Order Bag

Syntax

ORDDESTROY(<cOrderName> [, <cOrderBagName>]) --> NIL

Arguments

<cOrderName> Name of the order to remove

<cOrderBagName> Name of the order bag from which order id to be removed

Returns

ORDDESTROY() always returns NIL.

Description

This function attempts to remove the order named <cOrderName> from the file containing the order bag name <cOrderBagName>. If <cOrderBagName> is not specified,then the name of the file will be based on the value of the ORDNAME() function.If the extension is not included with the name of the order file,then the extension will be obtained from the default extension of the current and active RDD.

The DBFNTX driver do not support multiple order bags;therefore,there cannot be an order to "destroy" from a bag.This function only works for those drivers with support multiple orders bags (e.q. DBFCDX and RDDADS drivers).

Examples

```
USE TEsts VIA "DBFCDX" NEW
ORDdestroy( "lName", "tests" )
```

Tests

See examples

Status

Started

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

ORDCREATE()

ORDFOR()

Return the FOR expression of an Order

Syntax

ORDFOR(<xOrder>[, <cOrderBagName>]) --> cForExp

<xOrder> It the name of the target order,or the numeric position of the order.

<cOrderBagName> Name of the order bag.

Returns

ORDFOR() returns a expression containing the FOR condition for an order.

Description

This function returns a character string that is the expression for the FOR condition for the specified order.The order may be specified if <xOrder> is the name of the order.However,<xOrder> may be an numeric which represent the position in the order list of the desired Order.

Examples

```
USE Tests NEW via _DBFCDX
INDEX ON  Tests->Id ;
   TO  TESTS          ;
   FOR Tests->Id > 100

ORDFOR( "Tests" )      // Returns: Tests->Id > 100
```

Tests

See examples

Status

Started

Compliance

This function is Ca-Clipper compliant with one exception. If the <xOrder> paramter is not specified or <xOrder> is 0, the current active order is used.

Platforms

All

Files

Library is rdd

See Also:

ORDKEY()
ORDCREATE()
ORDNAME()
ORDNUMBER()

ORDKEY()

Return the key expression of an Order

Syntax

ORDKEY(<cOrderName> | <nOrder> [, <cOrderBagName>]) --> cExpKey

Arguments

<xOrder> It the name of the target order,or the numeric position of the order.

<cOrderBagName> Name of the order bag.

Returns

<cExpKey> Returns a character string, cExpKey.

Description

ORDKEY() is an Order management function that returns a character expression, cExpKey, that represents the key expression of the specified Order.

You may specify the Order by name or with a number that represents its position in the Order List. Using the Order name is the preferred method.

The active RDD determines the Order capacity of an Order Bag. The default DBFNTX and the DBFNDX drivers only support single-Order Bags, while other RDDs may support multiple-Order Bags (e.g., the DBFCDX and DBFMDX drivers).

Examples

```
USE Customer NEW via _DBFCDX
INDEX ON  Customer->Acct  ;
   TO  Customer           ;
   FOR Customer->Acct > "AZZZZZ"
Index on Custumer->Id to Cusid

ORDKEY( "Customer" )      // Returns: Customer->Acct
Set order to 2
ORDKEY()                  // Returns: Custumer->Id
```

Status

Started

Compliance

This function is Ca-Clipper compliant with one exception. If the <xOrder> paramter is not specified or <xOrder> is 0, the current active order is used.

Platforms

All

Files

Library is rdd

See Also:

ORDFOR()
ORDNAME()
ORDNUMBER()
ORDKEY()

ORDLISTADD()

Add Orders to the Order List

Syntax

ORDLISTADD(<cOrderBagName>
[, <cOrderName>]) --> NIL

Arguments

<cOrderBagName> is the name of a disk file containing one or more Orders. You may specify <cOrderBagName> as the filename with or without the pathname or appropriate extension. If you do not include the extension as part of <cOrderBagName> HARBOUR uses the default extension of the current RDD.

<cOrderName> the name of the specific Order from the Order Bag to be added to the Order List of the current work area. If you do not specify <cOrderName>, all orders in the Order Bag are added to the Order List of the current work area.

Returns

ORDLISTADD() always returns NIL.

Description

ORDLISTADD() is an Order management function that adds the contents of an Order Bag , or a single Order in an Order Bag, to the Order List. This function lets you extend the Order List without issuing a SET INDEX command that, first, clears all the active Orders from the Order List.

Any Orders already associated with the work area continue to be active. If the newly opened Order Bag contains the only Order associated with the work area, it becomes the controlling Order; otherwise, the controlling Order remains unchanged.

After the new Orders are opened, the work area is positioned to the first logical record in the controlling Order.

ORDLISTADD() is similar to the SET INDEX command or the INDEX clause of the USE command, except that it does not clear the Order List prior to adding the new order(s).

ORDLISTADD() supersedes the DBSETINDEX() function.

The active RDD determines the Order capacity of an Order Bag. The default DBFNTX and the DBFNDX drivers only support single-Order Bags, while other RDDs may support multiple-Order Bags (e.g., the DBFCDX and DBPX drivers). When using RDDs that support multiple Order Bags, you must explicitly SET ORDER (or ORDSETFOCUS()) to the desired controlling Order. If you do not specify a controlling Order, the data file will be viewed in natural Order.

Examples

```
In this example Customer.cdx contains three orders, CuAcct,
 CuName, and CuZip.  ORDLISTADD() opens Customer.cdx but only uses the
 order named CuAcct:

 USE Customer VIA "DBFCDX" NEW
 ORDLISTADD( "Customer", "CuAcct" )
```

Tests

Status

Started

All

Files

Library is rdd

See Also:

ARRAY( )

ORDLISTCLEAR()

Clear the current Order List

Syntax

ORDLISTCLEAR() --> NIL

Arguments

Returns

ORDLISTCLEAR() always returns NIL.

Description

ORDLISTCLEAR() is an Order management function that removes all Orders from the Order List for the current or aliased work area. When you are done, the Order List is empty.

This function supersedes the function DBCLEARINDEX().

```
USE Sales NEW
SET INDEX TO SaRegion, SaRep, SaCode
.
. < statements >
.
ORDLISTCLEAR()      // Closes all the current indexes
```

Tests

Status

Started

All

Files

Library is rdd

See Also:

ARRAY()

ORDLISTREBUILD()

Rebuild all Orders in the Order List of the current work area

Syntax

ORDLISTREBUILD() --> NIL

Arguments

Returns

ORDLISTREBUILD() always returns NIL.

Description

ORDLISTREBUILD() is an Order management function that rebuilds all the orders in the current or aliased Order List.

To only rebuild a single Order use the function ORDCREATE().

Unlike ORDCREATE(), this function rebuilds all Orders in the Order List. It is equivalent to REINDEX.

```
USE Customer NEW
SET INDEX TO CuAcct, CuName, CuZip
ORDLISTREBUILD()     // Causes CuAcct, CuName, CuZip to
                     // be rebuilt
```

Tests

Status

Started

All

Files

Library is rdd

See Also:

ORDCREATE()

ORDNAME()

Return the name of an Order in the Order List

Syntax

ORDNAME(<nOrder>[,<cOrderBagName> --> cOrderName

Arguments

<nOrder> is an integer that identifies the position in the Order List of the target Order whose database name is sought.

<cOrderBagName> is the name of a disk file containing one or more Orders. You may specify <cOrderBagName> as the filename with or without the pathname or appropriate extension. If you do not include the extension as part of <xcOrderBagName> HARBOUR uses the default extension of the current RDD.

Returns

ORDNAME() returns the name of the specified Order in the current Order List or the specified Order Bag if opened in the Current Order list.

Description

ORDNAME() is an Order management function that returns the name of the specified Order in the current Order List.

If <cOrderBagName> is an Order Bag that has been emptied into the current Order List, only those Orders in the Order List that correspond to <cOrderBagName> Order Bag are searched.

The active RDD determines the Order capacity of an Order Bag. The default DBFNTX and the DBFNDX drivers only support single-Order Bags, while other RDDs may support multiple-Order Bags (e.g., the DBFCDX and DBPX drivers).

Examples

```
This example retrieves the name of an Order using its position
 in the order list:

 USE Customer NEW
 SET INDEX TO CuAcct, CuName, CuZip
 ORDNAME( 2 )                        // Returns: CuName

This example retrieves the name of an Order given its position
 within a specific Order Bag in the Order List:

 USE Customer NEW
 SET INDEX TO Temp, Customer
 // Assume Customer contains CuAcct, CuName, CuZip
 ORDNAME( 2, "Customer" )            // Returns: CuName
```

Tests

Status

Started

All

Files

Library is rdd

See Also:

ORDFOR()
ORDKEY()
ORDNUMBER()

ORDNUMBER()

Return the position of an Order in the current Order List

Syntax

ORDNUMBER(<cOrderName> [, <cOrderBagName>]) --> nOrderNo

Arguments

<cOrderName> the name of the specific Order whose position in the Order List is sought.

<cOrderBagName> is the name of a disk file containing one or more Orders. You may specify <cOrderBagName> as the filename with or without the pathname or appropriate extension. If you do not include the extension as part of <cOrderBagName> HARBOUR uses the default extension of the current RDD.

Returns

the Order List.

Description

ORDNUMBER() is an Order management function that lets you determine the position in the current Order List of the specified Order. ORDNUMBER() searches the Order List in the current work area and returns the position of the first Order that matches <cOrderName>. If <cOrderBagName> is the name of an Order Bag newly emptied into the current Order List, only those orders in the Order List that have been emptied from <cOrderBagName> are searched.

If <cOrderName> is not found ORDNUMBER() raises a recoverable runtime error.

The active RDD determines the Order capacity of an Order Bag. The default DBFNTX driver only supports single-Order Bags, while other RDDs may support multiple-Order Bags (e.g., the DBFCDX and DBPX drivers).

Examples

```
USE Customer VIA "DBFNTX" NEW
SET INDEX TO CuAcct, CuName, CuZip
ORDNUMBER( "CuName" )            // Returns: 2
```

Tests

Status

Started

All

Files

Library is rdd

See Also:

INDEXORD()

ORDSETFOCUS()

Set focus to an Order in an Order List

Syntax

ORDSETFOCUS([<cOrderName> | <nOrder>]
[,<cOrderBagName>]) --> cPrevOrderNameInFocus

<cOrderName> is the name of the selected Order, a logical ordering of a database. ORDSETFOCUS() ignores any invalid values of <cOrderName>.

<nOrder> is a number representing the position in the Order List of the selected Order.

<cOrderBagName> is the name of a disk file containing one or more Orders. You may specify <cOrderBagName> as the filename with or without the pathname or appropriate extension. If you do not include the extension as part of <cOrderBagName> HARBOUR uses the default extension of the current RDD.

Returns

ORDSETFOCUS() returns the Order Name of the previous controlling Order.

Description

ORDSETFOCUS() is an Order management function that returns the Order Name of the previous controlling Order and optionally sets the focus to an new Order.

If you do not specify <cOrderName> or <nOrder>, the name of the currently controlling order is returned and the controlling order remains unchanged.

All Orders in an Order List are properly updated no matter what <cOrderName> is the controlling Order. After a change of controlling Orders, the record pointer still points to the same record.

The active RDD determines the Order capacity of an Order Bag. The default DBFNTX driver only supports single-Order Bags, while other RDDs may support multiple-Order Bags (e.g., the DBFCDX and DBPX drivers).

ORDSETFOCUS() supersedes INDEXORD().

Examples

```
USE Customer VIA "DBFNTX" NEW
SET INDEX TO CuAcct, CuName, CuZip
? ORDSETFOCUS( "CuName" )        // Displays: "CuAcct"
? ORDSETFOCUS()                  // Displays: "CuName"
```

Status

Started

All

Files

Library is rdd

INDEXEXT()

Returns the file extension of the index module used in an application

Syntax

INDEXEXT() --> <cExtension>

Arguments

Returns

<cExtension> Current driver file extension

Description

This function returns a string that tells what indexes are to be used or will be created in the compiled application.The default value is ".NTX". This is controled by the particular database driver that is linked with the application,.

Examples

```
IF INDEXEXT()==".NTX"
    ? "Current driver being used is DBFNTX"
Endif
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

INDEXKEY()
INDEXORD()

INDEXKEY()

Yields the key expression of a specified index file.

Syntax

INDEXKEY(<nOrder>) --> <cIndexKey>

Arguments

<nOrder> Index order number

Returns

<cIndexKey> The index key

Description

This function returns a character string stored in the header of the index file

The index key is displayed for an index file that is designated by <nOrder>,its position in the USE...INDEX or SET INDEX TO command in the currently selected or designated work area.If there is no corresnponding index key at the specified order position,a NULL byte will be returned.

Examples

```
USE TESTS NEW INDEX TEST1
? INDEXKEY(1)
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

INDEXORD()

INDEXORD()

Returns the numeric position of the controlling index.

Syntax

INDEXORD() --> <nPosition>

Arguments

Returns

<nPosition> Ordinal position of a controling index

Description

The INDEXORD() function returns the numeric position of the current controlling index in the selected or designated work area. A returned value of 0 indicated that no active index is controlling the database,which therefore is in the natural order.

Examples

```
USE TESTS NEW INDEX TEST1
IF INDEXORD()>0
    ? "Current order is ",INDEXORD()
Endif
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rdd

See Also:

INDEXKEY()

OVERVIEW

HARBOUR Read me

Description

The Harbour project

```
**********************************************************************
*  This file contains information on obtaining, installing, and using  *
*  Harbour. Please read it *completely* before asking for help.        *
**********************************************************************
```

Harbour is a free implementation of an xBase language compiler. It is designed to be source code compatible with the CA-Clipper(r) compiler. That means that if you've got some code that would compile using CA-Clipper(r) then it should compile under Harbour. The Harbour-Project web page is:
http://www.Harbour-Project.org/

Status and other information is always available from the web site. There is a Harbour mailing list. Harbour is still at a very early stage of development, so the mailing list is very much a Developers only list, although every body is welcome to join in the discussions.

We would like you to join the Harbour development team. If you are interested you may suscribe to our mailing list and start contributing to this free public project.

Please feel free to report all questions, ideas, suggestions, fixes, code, etc. you may need and want. With the help of all of you, the Harbour compiler and runtime libraries will become a reality very soon.

What this distribution contains
===============================

This distribution is a Source code only distribution. It does not contain any executable files. Executable versions of Harbour are available from the web site. Executable versions of Harbour DO NOT create runable programs. Harbour at the moment produces C output code, which must be compiled with the Harbour Virtual Machine and the support libraries in order to create a functioning program. Please test running Harbour against your Clipper source code and report any problems that might occur.

Very important: The preprocessor functionality is now working.

Installation
----

1. Unzip with Harbour zip file using pkunzip or equivalent.
E.G. pkunzip -d build72.zip
This will create Harbour/ directory and all the relevant sub directories.

2. Compile Harbour using your C compiler. Make files for different platforms are included in the <WHERE ARE THEY?> directory.

--- COPYRIGHT ---

What copyright information do we have

--- LICENCE ---

Information about the License for usage of Harbour is available in the file LICENCE.TXT (when we have a license)

--- DISCLAIMER ---

Participants of The Harbour Project assume no responsibility for errors or omissions in these materials.

THESE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

See Also:

License

@...Get

Creates a GET object and displays it to the screen

Syntax

```
@ <nRow>,<nCol> [SAY <cSay> [PICTURE <cSayPict>] COLOR <cSayColor> ]
GET <xVar> [PICTURE <cGetPict>] [WHEN <lWhen>] [COLOR <cGetColor>]
[VALID <lValid> / RANGE <xStart>,<xEnd>]
```

Arguments

<nRow>	The row coordinate.
<nCol>	The column coordinate.
<cSay>	Message to display.
<cSayPict>	Character expression of PICTURE displayed.
<cSayColor>	Color to be Used for the SAY expression.
<xVar>	An variable/field name.
<cGetPict>	Character expression of PICTURE to get.
<lWhen>	Logical expression to allow GET.
<lValid>	Logical expression to validate GET input.
<xStart>	Lower RANGE value.
<xEnd>	Upper RANGE value.
<cGetColor>	Color string to be used for the GET expression.

Returns

Description

This command adds a GET object to the reserved array variable named GETLIST[] and displays it to the screen. The field or variable to be added to the GET object is specified in <xVar> and is displayed at row, column coordinate <nRow>, <nCol>.

If the SAY clause is used <cSay> will be displayed starting at <nRow>,<nCol>, with the field variable <xVar> displayed at ROW(), COL()+ 1. If <cSayPicr>, the picture template for the SAY expression <cSay>, is used, all formatting rules contained will apply See the TRANSFORM I function for futher information.

If <cGetPict> is specified, the PICTURE clause of <xVar> will be used for the GET object and all formatting rules will apply. See the table below for GET formatting rules.

If the WHEN clause is specified,when <lWhen> evaluates to a logical true (.T.) condition, the GET object will he activated otherwise the GET object will be skipped and no information will be obtained via the screen. The name of a user-defined function returning a logical true (.T.) or false (F.) or a code block may be ,specified in <lWhen> This clause not activated until a READ command or READMODAL() function call is issued.

If the VALID clause is specified and <lValid> evaluates to it logical true (.T.) condition the current GET will be considered valid and the get operation will continue onto the next active GET object. If not, the cursor will remain on this GET object until aborted or until the condition in <lValid> evaluates to true (.T.). The name of a user-defined function returning a logical true (.T.) or false (.F.) or it code block may be specified in <lValid>. This clause is not activated until a READ command or READMODAL( ) function call is issued.

If the RANGE clause is specified instead of the VALID clause, the two inclusive range values for <xVar> must be specified in <xStart> and <xEnd>. Id <xVar> is a date data type,<xStart> and <xEnd> must also be date data types; if <xVar> is a numeric data type <xStart> and <xEnd> must also be numeric data types. If a value fails the RANGE test ,a message of OUT OF RANGE will appear in the SCOREBOARD area (row = 0, col = 60).The RANGE message may be turned off it the SET SCOREBOARD command or SET() function appropriately toggled.

NOTE GET functions/formatting rules:

@A	Allows only alphabetic characters.
@B	Numbers will be left justified
@C	All positive numbers will be followes by CR.
@D	All dates will be in the SET DATE format.
@E	Dates will be in British formal: numbers in European format.
@K	Allows a suggested value to be seen within the GET
	area but clears It if any noncu sor key is pressed when
	the cursor is in the first Position in the GET area.
@R	Nontemplate characters will be inserted.
@S<nSize>	Allows horizontal scrolling of a field or variable that
	is <nSize> characters wide.
@X	All negative numbers will be followed by DB
@Z	Displays zero values as blanks.
@!	Forces uppercase lettering
@(	Displays negative numbers in parentheses with leading spaces.
@)	Displays negative numbers in parentheses without leading spaces.

GET templates/formatting rules:

A	Only alphabetic characters allowed.
N	Only alphabetic and numeric characters allowed
X	Any character allowed.
L	Only T or F allowed For logical data.
Y	Only or N allowed for logical data.
9	Only digits, including signs, will be allowed.
#	Only digits, signs. and spaces will he allowed.
!	Alphabetic characters are converted to Uppercase.
$	Dollar will be displayed in place of leading
	spaces for numeric data types.
*	Asterisk,, will Be displayed in place of leading spaces
	for numeric data types.
.	Position of decimal point
,	Position of comma.

Format PICTURE functions may he grouped together as well as used in Conjunction with a PICTURE templates;however, a blank space must be included in the PICTURE string if there are both functions and templates.

Examples

```
Function Main()
Local cVar:=Space(50)
Local nId:=0
cls
@ 3,1 SAY "Name" GET cVar PICTURE "@!S 30"
@ 4,1 SAY "Id"   GET nId  PICTURE "999.999"
READ
? "The name you entered is",cVar
? "The id you entered is",nId
RETURN NIL
```

Tests

See Examples

Status

Ready

Compliance

This command is Ca-Clipper compatible

Platforms

All

See Also:

- @...SAY
- ARRAY()
- TRANSFORM()

@...SAY

Displays data to specified coordinates of the current device.

Syntax

@ <nRow>,<nCol> SAY <xValue> [PICTURE <cPict>] [COLOR <cColor>]

Arguments

<nRow>	Row coordinate
<nCol>	Column coordinate
<xValue>	Value to display
<cPict>	PICTURE format
<cColor>	Color string

Returns

Description

This command displays the contents of <xValue> at row column coordinates <nRow>, <nCol>. A PICTURE clause may be speclfied in <cPict>. If the current device is set to the printer, the output will go to the printer; the default is for all output to go to the screen.

For a complete list of PICTURES templates and functions, see the @...GET command.

Examples

```
Function Main
Cls
@ 2,1 SAY "Harbour"
@ 3,1 SAY "is" COLOR "b/r+"
@ 4,1 SAY "Power" PICTURE "@!"
Return NIL
```

Tests

See Examples

Status

Ready

Compliance

This command is Ca-Clipper compliant

Platforms

All

Files

See Also:

@...Get
SET DEVICE
TRANSFORM()

__SETCENTURY()

Set the Current Century

Syntax

__SETCENTURY([<lFlag> | <cOnOff>]) --> lPreviousValue

Arguments

setting (4-digit years) .F. or "OFF" to disable the century setting (2-digit years)

Returns

Files

Library is rtl

SET()

Changes or evaluated enviromental settings

Syntax

Set(<nSet> [, <xNewSetting> [, <xOption>]]) --> xPreviousSetting

Arguments

<nSet> Set Number

<xNewSetting> Any expression to assing a value to the seting

<xOption> Logical expression

<nSet>	<xNewSetting>	<xOption>
_SET_ALTERNATE	<lFlag> | <cOnOff>	

file has been opened or created with _SET_ALTFILE. If disabled, which is the default, QOUT() and QQOUT() only write to the screen (and/or to the PRINTFILE). Defaults to disabled.

_SET_ALTFILE <cFileName> <lAdditive>

<lAdditive> is TRUE and the file already exists, the file is opened and positioned at end of file. Otherwise, the file is created. If a file is already opened, it is closed before the new file is opened or created (even if it is the same file). The default file extension is ".txt". There is no default file name. Call with an empty string to close the file.

_SET_AUTOPEN <lFlag> | <cOnOff>

_SET_AUTORDER <lFlag> | <cOnOff>

_SET_AUTOSHARE <lFlag> | <cOnOff>

_SET_BELL <lFlag> | <cOnOff>

when a GET validation fails. Disabled by default.

_SET_CANCEL <lFlag> | <cOnOff>

program. When disabled, both keystrokes can be read by INKEY(). Note: SET KEY has precedence over SET CANCEL.

_SET_COLOR <cColorSet>

"<standard>,<enhanced>,<border>,<background>, <unselected>". Each color pair uses the format "<foreground>/<background>". The color codes are space or "N" for black, "B" for blue, "G" for green, "BG" for Cyan, "R" for red, "RB" for magenta, "GR" for brown, "W" for white, "N+" for gray, "B+" for bright blue, "G+" for bright green, "BG+" for bright cyan, "R+" for bright red, "RB+" for bright magenta, "GR+" for yellow, and "W+" for bright white. Special codes are "I" for inverse video, "U" for underline on a monochrome monitor (blue on a color monitor), and "X" for blank. The default color is "W/N,N/W,N,N,N/W".

_SET_CONFIRM <lFlag> | <cOnOff>

default, typing past the end will leave a GET.

_SET_CONSOLE <lFlag> | <cOnOff>

disabled, screen output is suppressed (Note: This setting does not affect OUTSTD() or OUTERR()).

_SET_CURSOR <nCursorType>

the screen cursor is hidden.

_SET_DATEFORMAT <cDateFormat>

to American ("mm/dd/yy"). Other formats include ANSI ("yy.mm.dd"), British ("dd/mm/yy"), French ("dd/mm/yy"), German ("dd.mm.yy"), Italian ("dd-mm-yy"), Japan ("yy/mm/dd"), and USA ("mm-dd-yy"). SET CENTURY modifies the date format. SET CENTURY ON replaces the "y"s with "YYYY". SET CENTURY OFF replaces the "y"s with "YY".

_SET_DEBUG <lStatus>

the default, Alt+D can be read by INKEY(). (Also affected by AltD(1) and AltD(0))

_SET_DECIMALS <nNumberOfDecimals>

when SET FIXED is ON. Defaults to 2. If SET FIXED is OFF, then SET DECIMALS is only used to determine the number of decimal digits to use after using EXP(), LOG(), SQRT(), or division. Other math operations may adjust the number of decimal digits that the result will display. Note: This never affects the precision of a number. Only the display format is affected.

_SET_DEFAULT <cDefaultDirectory>

to current directory (blank).

_SET_DELETED <lFlag> | <cOnOff>

deleted records will be ignored.

_SET_DELIMCHARS <cDelimiters>

_SET_DELIMITERS <lFlag> | <cOnOff>

delimiters are used.

_SET_DEVICE <cDeviceName>

to the printer device or file set by _SET_PRINTFILE. When set to anything else, all output is sent to the screen. Defaults to "SCREEN".

_SET_EPOCH <nYear>

2-digit year is greater than or equal to the year part of the epoch, the century part of the epoch is added to the year. When a 2-digit year is less than the year part of the epoch, the century part of the epoch is incremented and added to the year. The default epoch is 1900, which converts all 2-digit years to 19xx. Example: If the epoch is set to 1950, 2-digit years in the range from 50 to 99 get converted to 19xx and 2-digit years in the range 00 to 49 get converted to 20xx.

_SET_ESCAPE <lFlag> | <cOnOff> *

pressing Esc during a READ is ignored, unless the Esc key has been assigned to a function using SET KEY.

_SET_EVENTMASK <nEventCodes>

events. INKEY_LDOWN allows the left mouse button down click. INKEY_LUP allows the left mouse button up click. INKEY_RDOWN allows the right mouse button down click. INKEY_RUP allows the right mouse button up clock. INKEY_KEYBOARD allows keyboard keystrokes. INKEY_ALL allows all of the preceding events. Events may be combined (e.g., using INKEY_LDOWN + INKEY_RUP will allow left mouse button down clicks and right mouse button up clicks). The default is INKEY_KEYBOARD.

_SET_EXACT <lFlag> | <cOnOff>

checking for equality. When disabled, which is the default, all string comparisons other than "==" treat two strings as equal if the right hand string is "" or if the right hand string is shorter than or the same length as the left hand string and all of the characters in the right hand string match the corresponding characters in the left hand string.

_SET_EXCLUSIVE <lFlag> | <cOnOff>

mode. When disabled, all database files are opened in shared mode. Note: The EXCLUSIVE and SHARED clauses of the USE command can be used to override this setting.

_SET_EXIT <lFlag> | <cOnOff>

enables them as exit keys, and false (.F.) disables them. Used internally by the ReadExit() function.

_SET_EXTRA <lFlag> | <cOnOff>

_SET_EXTRAFILE <cFileName> <lAdditive>

<lAdditive> is TRUE and the file already exists, the file is opened and positioned at end of file. Otherwise, the file is created. If a file is already opened, it is closed before the new file is opened or created (even if it is the same file). The default file extension is ".prn". There is no default file name. Call with an empty string to close the file.

_SET_FIXED <lFlag> | <cOnOff>

decimal digits set by SET DECIMALS, unless a PICTURE clause is used. When disabled, which is the default, the number of decimal digits that are displayed depends upon a variety of factors. See _SET_DECIMALS for more.

_SET_INSERT <lFlag> | <cOnOff>

which is the default, characters typed in a GET or MEMOEDIT overwrite. Note: This setting can also be toggled between on and off by pressing the Insert key during a GET or MEMOEDIT.

_SET_INTENSITY <lFlag> | <cOnOff>

enhanced color setting. When disabled, GETs and PROMPTs are displayed using the standard color setting.

_SET_LANGUAGE <cLanguageID>

_SET_MARGIN <nColumns>

reflects the printer's column position including the margin (e.g., SET MARGIN TO 5 followed by DEVPOS(5, 10) makes PCOL() return 15).

_SET_MBLOCKSIZE <nMemoBlockSize>

_SET_MCENTER <lFlag> | <cOnOff>

default, display PROMPTS at column position 0 on the MESSAGE row.

_SET_MESSAGE <nRow>

PROMPTs are displayed on the set row. Note: It is not possible to display prompts on the top-most screen row, because row 0 is reserved for the SCOREBOARD, if enabled.

_SET_MFILEEXT <cMemoFileExt>

_SET_OPTIMIZE <lFlag> | <cOnOff>

_SET_PATH <cDirectories>

located in the DEFAULT directory. Defaults to no path (""). Directories must be separated by a semicolon (e.g., "C:\DATA;C:\MORE").

_SET_PRINTER <lFlag> | <cOnOff>

file has been opened or created with _SET_ALTFILE. If disabled, which is the default, QOUT() and QQOUT() only write to the screen (and/or to the ALTFILE).

_SET_PRINTFILE <cFileName> <lAdditive>

<lAdditive> is TRUE and the file already exists, the file is opened and positioned at end of file. Otherwise, the file is created. If a file is already opened, it is closed before the new file is opened or created (even if it is the same file). The default file extension is ".prn". The default file name is "PRN", which maps to the default printer device. Call with an empty string to close the file.

_SET_SCOREBOARD <lFlag> | <cOnOff>

screen row 0. When disabled, READ and MEMOEDIT status messages are suppressed.

_SET_SCROLLBREAK <lFlag> | <cOnOff>

_SET_SOFTSEEK <lFlag> | <cOnOff>

that is higher than the sought after key or to LASTREC() + 1 if there is no higher key. When disabled, which is the default, a SEEK that fails will position the record pointer to LASTREC()+1.

_SET_STRICTREAD <lFlag> | <cOnOff>

_SET_TYPEAHEAD <nKeyStrokes>

and the maximum is 4096.

_SET_UNIQUE <lFlag> | <cOnOff>

indexes are allowed duplicate keys.

_SET_VIDEOMODE <nValue>

_SET_WRAP <lFlag> | <cOnOff>

and from the first position to the last. When disabled, which is the default, there is a hard stop at the first and last positions.

Returns

SET() The current or previous setting

Files

Library is rtl

__SetFunction()

Assign a character string to a function key

Syntax

```
__SetFunction( <nFunctionKey>, [<cString>] ) --> NIL
```

Arguments

<nFunctionKey> is a number in the range 1..40 that represent the function key to be assigned.

<cString> is a character string to set. If is not specified, the function key is going to be set to NIL releasing by that any previous __SetFunction() or SETKEY() for that function.

Returns

__SetFunction() always return NIL.

Description

__SetFunction() assign a character string with a function key, when this function key is pressed, the keyboard is stuffed with this character string. __SetFunction() has the effect of clearing any SETKEY() previously set to the same function number and vice versa.

nFunctionKey	Key to be set
1 .. 12	F1 .. F12
13 .. 20	Shift-F3 .. Shift-F10
21 .. 30	Ctrl-F1 .. Ctrl-F10
31 .. 40	Alt-F1 .. Alt-F10

SET FUNCTION command is preprocessed into __SetFunction() function during compile time.

Examples

```
// Set F1 with a string
CLS
__SetFunction( 1, "I Am Lazy" + CHR( 13 ) )
cTest := SPACE( 20 )
@ 10, 0 SAY "type something or F1 for lazy mode " GET cTest
READ
? cTest
```

Status

Ready

Compliance

Harbour use 11 and 12 to represent F11 and F12, while CA-Clipper use 11 and 12 to represent Shift-F1 and Shift-F2.

Platforms

All

Files

Library is rtl

See Also:

INKEY()
SETKEY()
__KEYBOARD()

SET KEY

SET FUNCTION

Assign a character string to a function key

Syntax

SET FUNCTION <nFunctionKey> TO [<cString>]

Arguments

<nFunctionKey> is a number in the range 1..40 that represent the function key to be assigned.

<cString> is a character string to set. If is not specified, the function key is going to be set to NIL releasing by that any previous Set Function or SETKEY() for that function.

Description

Set Function assign a character string with a function key, when this function key is pressed, the keyboard is stuffed with this character string. Set Function has the effect of clearing any SETKEY() previously set to the same function number and vice versa.

nFunctionKey	Key to be set
1 .. 12	F1 .. F12
13 .. 20	Shift-F3 .. Shift-F10
21 .. 30	Ctrl-F1 .. Ctrl-F10
31 .. 40	Alt-F1 .. Alt-F10

SET FUNCTION command is preprocessed into __SetFunction() function during compile time.

Examples

```
// Set F1 with a string
CLS
Set Function  1 to  "I Am Lazy" + CHR( 13 )
cTest := SPACE( 20 )
@ 10, 0 SAY "type something or F1 for lazy mode " GET cTest
READ
? cTest
```

Status

Ready

Compliance

Harbour use 11 and 12 to represent F11 and F12, while CA-Clipper use 11 and 12 to represent Shift-F1 and Shift-F2.

Platforms

All

See Also:

INKEY()
SETKEY()
__KEYBOARD()

SETKEY()

Assign an action block to a key

Syntax

SETKEY(<anKey> [, <bAction> [, <bCondition>]])

Arguments

<anKey> is either a numeric key value, or an array of such values

<bAction> is an optional code-block to be assigned

<bCondition> is an optional condition code-block

Returns

Description

The SetKey() function returns the current code-block assigned to a key when called with only the key value. If the action block (and optionally the condition block) are passed, the current block is returned, and the new code block and condition block are stored. A group of keys may be assigned the same code block/condition block by using an array of key values in place on the first parameter.

Examples

```
local bOldF10 := setKey( K_F10, {|| Yahoo() } )
... // some other processing
SetKey( K_F10, bOldF10 )
... // some other processing
bBlock := SetKey( K_SPACE )
if bBlock != NIL ...

// make F10 exit current get, but only if in a get - ignores other
// wait-states such as menus, achoices, etc...
SetKey( K_F10, {|| GetActive():State := GE_WRITE },;
 {|| GetActive() != NIL } )
```

Tests

None definable

Status

Ready

Compliance

SETKEY() is mostly CA-Clipper compliant. The only difference is the addition of the condition code-block parameter, allowing set-keys to be conditionally turned off or on. This condition-block cannot be returned once set - see SetKeyGet()

Files

Library is rtl

See Also:

HB_SETKEYSAVE()

HB_SetKeyGet()

Determine a set-key code block & condition-block

Syntax

HB_SETKEYGET(<nKey> [, <bConditionByRef>])

Arguments

<anKey> is an numeric key value

<bConditionByRef> is an optional return-parameter

Returns

Description

The HB_SetKeyGet() function returns the current code-block assigned to a key, and optionally assignes the condition-block to the return-parameter

Examples

```
local bOldF10, bOldF10Cond
bOldF10 := HB_SetKeyGet( K_F10, @bOldF10Cond )
... // some other processing
SetKey( K_F10, bOldF10, bOldF10Cond )
```

Tests

See test code above

Status

Ready

Compliance

HB_SETKEYGET() is a new function and hence not CA-Clipper compliant.

Files

Library is rtl

See Also:

SETKEY()
HB_SETKEYSAVE()
HB_SetKeyCheck()

HB_SETKEYSAVE()

Returns a copy of internal set-key list, optionally overwriting

Syntax

HB_SETKEYSAVE([<OldKeys>])

Arguments

<OldKeys> is an optional set-key list from a previous call to HB_SetKeySave(), or NIL to clear current set-key list

Returns

Description

HB_SetKeySave() is designed to act like the set() function which returns the current state of an environment setting, and optionally assigning a new value. In this case, the "environment setting" is the internal set-key list, and the optional new value is either a value returned from a previous call to SetKeySave() - to restore that list, or the value of NIL to clear the current list.

Examples

```
local aKeys := HB_SetKeySave( NIL )  // removes all current set=keys
... // some other processing
HB_SetKeySave( aKeys )
```

Tests

None definable

Status

Ready

Compliance

HB_SETKEYSAVE() is new.

Files

Library is rtl

See Also:

SETKEY()

HB_SetKeyCheck()

Impliments common hot-key activation code

Syntax

HB_SetKeyCheck(<nKey> [, <p1>][, <p2>][, <p3>])

Arguments

<nKey> is a numeric key value to be tested code-block, if executed

<p1>..<p3> are optional parameters that will be passed to the code-block

Returns

False If there is a hot-key association (before checking any condition): - if there is a condition-block, it is passed one parameter - <nKey> - when the hot-key code-block is called, it is passed 1 to 4 parameters, depending on the parameters passed to HB_SetKeyCheck(). Any parameters so passed are directly passed to the code-block, with an additional parameter being <nKey>

Description

HB_SetKeyCheck() is intended as a common interface to the SetKey() functionality for such functions as ACHOICE(), DBEDIT(), MEMOEDIT(), ACCEPT, INPUT, READ, and WAIT

Examples

```
// within ReadModal()
if HB_SetKeyCheck( K_ALT_X, GetActive() )
... // some other processing
endif
// within TBrowse handler
case HB_SetKeyCheck( nInkey, oTBrowse )
  return
case nInKey == K_ESC
... // some other processing
```

Tests

None definable

Status

Ready

Compliance

HB_SETKEYCHECK() is new.

Files

Library is rtl

See Also:

SETKEY()
HB_SETKEYSAVE()

SET KEY

Assign an action block to a key

Syntax

SET KEY <anKey> to p<bAction>] [when <bCondition>])

Arguments

<anKey> is either a numeric key value, or an array of such values

<bAction> is an optional code-block to be assigned

<bCondition> is an optional condition code-block

Description

The Set Key Command function is translated to the SetKey() function witch returns the current code-block assigned to a key when called with only the key value. If the action block (and optionally the condition block) are passed, the current block is returned, and the new code block and condition block are stored. A group of keys may be assigned the same code block/condition block by using an array of key values in place on the first parameter.

Examples

```
local bOldF10 := setKey( K_F10, {|| Yahoo() } )
... // some other processing
Set Key  K_F10 to  bOldF10)
... // some other processing
bBlock := SetKey( K_SPACE )
if bBlock != NIL ...

// make F10 exit current get, but only if in a get - ignores other
// wait-states such as menus, achoices, etc...
SetKey( K_F10, {|| GetActive():State := GE_WRITE },;
 {|| GetActive() != NIL } )
```

Tests

None definable

Status

Ready

Compliance

SET KEY is mostly CA-Clipper compliant. The only difference is the addition of the condition code-block parameter, allowing set-keys to be conditionally turned off or on. This condition-block cannot be returned once set - see SetKeyGet()

See Also:

HB_SETKEYSAVE()

SETTYPEAHEAD()

Sets the typeahead buffer to given size.

Syntax

SETTYPEAHEAD(<nSize>) --> <nPreviousSize>

Arguments

<nSize> is a valid typeahead size.

Returns

<nPreviousSize> The previous state of _SET_TYPEAHEAD

Description

This function sets the typeahead buffer to a valid given size as is Set(_SET_TYPEAHEAD) where used.

Examples

```
// Sets typeahead to 12
SetTypeahead( 12 )
```

Status

Ready

Compliance

SETTYPEAHEAD() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ARRAY()
__INPUT()

__XHELP()

Looks if a Help() user defined function exist.

Syntax

```
__XHELP() --> <xValue>
```

Arguments

Returns

Description

This is an internal undocumented Clipper function, which will try to call the user defined function HELP() if it's defined in the current application. This is the default SetKey() handler for the F1 key.

Status

Ready

Compliance

__XHELP() is fully CA-Clipper compliant.

Files

Library is rtl

SET DEFAULT

Establishes the Harbour search drive and directory.

Syntax

```
SET DEFAULT TO [<cPath>]
```

Arguments

<cPath> Drive and/or path.

Description

This command changes the drive and directory used for reading and writting database,index,memory, and alternate files.Specifying no parameters with this command will default the operation to the current logged drive and directory.

Examples

```
SET DEFAULT to c:\TEMP
```

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

SET PATH
CURDIR()
SET()

SET WRAP

Toggle wrapping the PROMPTs in a menu.

Syntax

```
SET WRAP on | OFF | (<lWrap>
```

Arguments

<lWrap> Logical expression for toggle

Description

This command toggles the highlighted bars in a @...PROMPT command to wrap around in a bottom-to-top and top-to-bottom manner.If the value of the logical expression <lWrap> is a logical false (.F.), the wrapping mode is set OFF;otherwise,it is set ON.

Examples

See Tests/menutest.prg

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

@...PROMPT
MENU TO

SET MESSAGE

Extablishes a message row for @...PROMPT command

Syntax

```
SET MESSAGE TO [<nRow> [CENTER]]
```

Arguments

<nRow> Row number to display the message

Description

This command is designed to work in conjuntion with the MENU TO and @...PROMPT commands.With this command, a row number between 0 and MAXROW() may be specified in <nRow>.This establishes the row on witch any message associated with an @...PROMPT command will apear.

If the value of <nRow> is 0,all messages will be supressed. All messaged will be left-justifies unless the CENTER clause is used.In this case,the individual messages in each @...PROMPT command will be centered at the designated row (unless <nRow> is 0).All messages are independent;therefor,the screen area is cleared out by the centered message will vary based on the length of each individual message.

Specifying no parameters with this command set the row value to 0, witch suppresses all messages output. The British spelling of CENTRE is also supported.

Examples

See Tests/menutest.prg

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

SET()
SET WRAP
@...PROMPT
MENU TO

SET PATH

Specifies a search path for opening files

Syntax

SET PATH TO [<cPath>]

Arguments

<cPath> Search path for files

Description

This command specifies the search path for files required by most commands and functions not found in the current drive and directory. This pertains primarily,but not exclusively, to databases,indexes, and memo files,as well as to memory,labels,and reports files. The search hirarchy is: 1 Current drive and directory,2 The SET DEFAULT path;3 The SET PATH path.

Examples

```
SET PATH TO c:\Harbour\Test
```

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

SET DEFAULT
CURDIR()
SET()

SET INTENSITY

Toggles the enhaced display of PROMPT's and GETs.

Syntax

```
SET INTENSITY  ON | off | (<lInte>)
```

Arguments

<lInte> Logical expression for toggle command

Description

This command set the field input color and @...PROMPT menu color to either highlighted (inverse video) or normal color. The default condition is ON (highlighted).

Examples

```
SET INTENSITY ON
```

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

@...Get
@...PROMPT
@...SAY
SET()

SET ALTERNATE

Toggle and echos output to an alternate file

Syntax

```
SET ALTERNATE to <cFile> [ADDITIVE]
SET ALTERNATE  on | OFF | (<lAlter>)
```

Arguments

<cFile> Name of alternate file.

<lAlter> Logical expression for toggle

Description

This command toggles and output console information to the alternate file <cFile>,provided that the command is toggled on or the condition <lAlter> is set to a logical true (.T.). If <cFile> does not has a file extension, .TXT will be assumed.The file name may optionally have a drive letter and/or directory path.If none is speficied, the current drive and directory will be used. If the ALTERNATE file is created but no ALTERNATE ON command is issued,nothing will be echoed to the file. If ADDITIVE clause is used,then the information will be appended to the existing alternate file.Otherwise,a new file will be created with the specified name (or an existing one will be overwritten) and the information will be appended to the file.The default is to create a new file. A SET ALTERNATE TO command will close the alternate file

Examples

```
SET ALTERNATE TO test.txt
SET ALTERNATE ON
? 'Harbour'
? "is"
? "Power"
SET ALTERNATE TO
SET ALTERNATE OFF
```

Status

Ready

Compliance

This command is Ca-Clipper Compliant.

See Also:

ARRAY()
SET PRINTER
SET CONSOLE
SET()

SET CENTURY

Toggle the century digits in all dates display

Syntax

```
SET CENTURY on | OFF | (<lCent>)
```

Arguments

<lCent> Logical expression for toggle

Description

This command allows the input and display of dates with the century prefix.It will be in the standart MM/DD/YYYY format unless specified by the SET DATE command or SET() function.If <lCent> is a logical true (.T.),the command will be set on;otherwise, the command will be set off

Examples

```
SET CENTURY ON
? DATE()
SET CENTURY OFF
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET DATE
SET EPOCH
CTOD()
DATE()
DTOC()
SET()

SET DATE

Assings a date format or chooses a predefined date data set.

Syntax

```
SET DATE FORMAT [TO] <cFormat>
SET DATE [TO] [ ANSI / BRITISH / FRENCH / GERMAN / ITALIAN / JAPAN
/ USA / AMERICAN]
```

Arguments

<cFormat> Keyword for date format

Description

This command sets the date format for function display purposes. If specified,<cFormat> may be a customized date format in which the letters d,m and y may be used to desing a date format.The default is an AMERICAN date format;specifying no parameters will set the date format to AMERICAN.Below is a table of the varius predefined dates formats.

Syntax	Date Format
ANSI	yy.mm.dd
BRITISH	dd/mm/yy
FRENCH	dd/mm/yy
GERMAN	dd.mm.yy
ITALIAN	dd-mm-yy
JAPAN	yy.mm.dd
USA	mm-dd-yy
AMERICAN	mm/dd/yy

Examples

```
SET DATE JAPAN
? DATE()
SET DATE GERMAN
? Date()
```

Tests

See tests/dates.prg

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET DATE
SET EPOCH
CTOD()
DATE()
DTOC()
SET()

SET EPOCH

Specifie a base year for interpreting dates

Syntax

```
SET EPOCH TO <nEpoch>
```

Arguments

<nEpoch> Base Century.

Description

This command sets the base year value for dates that have only two digits.The default setting is 1900.Dates between 01/01/0100 and 12/31/2999 are fully supported.

Examples

```
SET EPOCH TO 2000
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

- SET DATE
- SET CENTURY
- CTOD()
- DATE()
- DTOC()
- SET()

SET FIXED

Set the number of decimal position to be displayed

Syntax

```
SET FIXED on | OFF | (<lFixed>)
```

Arguments

<lFixed> Logical expression for toggle

Description

This command activates a system wide fixed placement of decimals places shown for all numeric outputs.If the value of <lFixed> is a logical true (.T.),FIXED will be turned ON;otherwise it will be turned OFF.

When SET DECIMALS OFF is used, the follow rules aply to the number of decimal placed displayed.

Addition	Same as operand with the greatest number of decimal digits
Subraction	Same as operand with the greatest number of decimal digits
Multiplication	Sum of operand decimal digits
Division	Determined by SET DECIMAL TO
Exponential	Determined by SET DECIMAL TO
LOG()	Determined by SET DECIMAL TO
EXP()	Determined by SET DECIMAL TO
SQRT()	Determined by SET DECIMAL TO
VAL()	Determined by SET DECIMAL TO

Examples

```
SET FIXED ON
? 25141251/362
SET FIXED OFF
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET DECIMALS
EXP()
LOG()
SQRT()
VAL()
SET()

SET PRINTER

Toggles the printer and controls the printer device

Syntax

```
SET PRINTER on | OFF
SET PRINTER (<lPrinter>)
SET PRINTER TO [<cPrinter>] [ADDITIVE]
```

Arguments

<lFixed> Logical condition by which to toggle the printer <cPrinter> A device name or an alternate name

Description

This command can direct all output that is not controled by the @...SAY command and the DEVPOS() and DEVOUT() functions to the printer.If specified,the condition <lPrinter> toggles the printer ON if a logical true (.T.) and OFF if a logical false (.F.).If no argument is specified in the command, the alternate file (if one is open) is closed, or the device is reselected and the PRINTER option is turned OFF.

If a device is specified in <cPrinter>, the outpur will be directed to that device instead of to the PRINTER.A specified device may be a literal string or a variable, as long as the variable is enclosed in parentheses.For a network,do not use a trailing colon when redirecting to a device.

If an alternate file is specified,<cPrinter> becomes the name of a file that will contain the output.If no file extension is specified an extension of.PRN will be defaulted to.

If the ADDITIVE clause is specified,the information will be appended to the end of the specified output file.Otherwise, a new file will be created with the specified name (or an existing file will first be cleared) and the information will then be appended to the file. The default is to create a new file.

Examples

```
SET PRINTER ON
SET PRINTER TO LPT1
? 25141251/362
SET PRINTER .F.
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET DEVICE
SET CONSOLE
ARRAY()
SET()

SET CONSOLE

Toggle the console display

Syntax

```
SET CONSOLE ON | off | (<lConsole>)
```

Arguments

<lConsole>	Logical expression for toggle command

Description

This command turns the screen display either off or on for all screens display other then direct output via the @...SAY commands or the <-> DEVOUT() function.

If <lConsole > is a logical true (.T.),the console will be turned ON;otherwise, the console will be turned off.

Examples

```
SET console on
? DATE()
SET console off
? date()
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET DEVICE
SET()

SET DECIMALS

Toggle the console display

Syntax

```
SET DECIMALS TO [<nDecimal>]
```

Arguments

<nDecimal>	Number of decimals places

Description

This command establishes the number of decimal places that Harbour will display in mathematical calculations,functions,memory variables, and fields.Issuing no parameter with this command will the default number of decimals to 0.For decimals to be seen,the SET FIXED ON command must be activated.

Examples

```
SET FIXED ON
? 25141251/362
SET DECIMALS TO 10
? 214514.214/6325
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET FIXED
SET()

SET DEVICE

Directs all @...SAY output to a device.

Syntax

```
SET DEVICE TO [printer | SCREEN ]
```

Arguments

Description

This command determines whether the output from the @...SAY command and the DEVPOS() and DEVOUT() function will be displayed on the printer.

When the device is set to the PRINTER,the SET MARGIN value adjusts the position of the column values accordingly.Also,an automatic page eject will be issued when the current printhead position is less than the last printed row.Finally,if used in conjunction with the @...GET commands,the values for the GETs will all be ignored.

Examples

```
SET DEVICE TO SCRENN
? 25141251/362
SET DEVICE TO PRINTER
SET PRINTER TO LPT1
? 214514.214/6325
SET PRINTER OFF
SET DEVICE TO SCREEN
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

[@...SAY]
[SET PRINTER]
[ARRAY()]
[SET()]

SET BELL

Toggle the bell to sound once a GET has been completed.

Syntax

```
SET BELL on | OFF | (<lBell>)
```

Arguments

<lBell> Logical expression for toggle command

Description

This command toggles the bell to sound whenever a character is entered into the last character positionof a GET,of if an invalid data type is entered into a GET.

If <lBell > is a logical true (.T.),the bell will be turned ON;otherwise, the belle will be turned off.

Examples

```
SET BEEL ON
cDummy:=space(20)
? 3,2 get cDummy
Read
SET bell off
```

Status

Ready

Compliance

This command is Ca-Clipper compliant

See Also:

SET()

SETMODE()

Change the video mode to a specified number of rows and columns

Syntax

SETMODE(<nRows>, <nCols>) --> lSuccess

Arguments

<nRows> is the number of rows for the video mode to set.

<nCols> is the number of columns for the video mode to set.

Returns

SETMODE() returns true if the video mode change was successful; otherwise, it returns false.

Description

SETMODE() is a function that change the video mode depend on the video card and monitor combination, to match the number of rows and columns specified. Note that there are only a real few combination or rows/cols pairs that produce the video mode change. The followings are availables for D.O.S:

12 rows x 40 columns	12 rows x 80 columns
25 rows x 40 columns	25 rows x 80 columns
28 rows x 40 columns	28 rows x 80 columns
50 rows x 40 columns	43 rows x 80 columns
	50 rows x 80 columns

The follow modes are avaliable to Windows

25 rows x 40 columns	25 rows x 80 columns
50 rows x 40 columns	43 rows x 80 columns
	50 rows x 80 columns

Some modes only are availables for color and/or VGA monitors. Any change produced on the screen size is updated in the values returned by MAXROW() and MAXCOL().

Examples

þ The first example change to a 12 lines of display mode:

```
IF SETMODE( 12, 40)
   ? "Hey man are you blind ?"
ELSE
   ? "Mom bring me my glasses!"
ENDIF
```

þ Next example change to a 50 lines mode:

```
IF SETMODE( 50, 80)
   ? "This wonderful mode was successfully set"
ELSE
   ? "Wait. this monitor are not made in rubber !"
ENDIF
```

Status

Ready

Compliance

Some of these modes are not availables on Clipper

Platforms

DOS,WIN32

Files

Source is gtdos.c,gtwin.c

See Also:

MAXCOL()
MAXROW()

ISALPHA()

Checks if leftmost character in a string is an alphabetic character

Syntax

ISALPHA(<cString>) --> lAlpha

Arguments

<cString> Any character string

Returns

Description

This function return a logical true (.T.) if the first character in <cString> is an alphabetic character.If not, the function will return a logical false (.F.).

Examples

```
QOUT( "isalpha( 'hello' ) = ", isalpha( 'hello' ) )
QOUT( "isalpha( '12345' ) = ", isalpha( '12345' ) )
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ISDIGIT()
ISLOWER()
ISUPPER()
LOWER()
UPPER()

ISDIGIT()

Checks if leftmost character is a digit character

Syntax

```
ISDIGIT( <cString> ) --> lDigit
```

Arguments

<cString> Any character string

Returns

Description

This function takes the caracter string <cString> and checks to see if the leftmost character is a digit,from 1 to 9.If so, the function will return a logical true (.T.);otherwise, it will return a logical false (.F.).

Examples

```
? ISDIGIT( '12345' )      // .T.
? ISDIGIT( 'abcde' )      // .F.
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ISALPHA()
ISLOWER()
ISUPPER()
LOWER()
UPPER()

ISUPPER()

Checks if leftmost character is an uppercased letter.

Syntax

```
ISUPPER( <cString> ) --> lUpper
```

Arguments

<cString> Any character string

Returns

Description

This function takes the caracter string <cString> and checks to see if the leftmost character is a uppercased letter.If so, the function will return a logical true (.T.);otherwise, it will return a logical false (.F.).

Examples

```
? ISUPPER( 'Abcde' )    // .T.
? ISUPPER( 'abcde' )    // .F.
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ISALPHA()
ISLOWER()
ISDIGIT()
LOWER()
UPPER()

ISLOWER()

Checks if leftmost character is an lowercased letter.

Syntax

```
ISLOWER( <cString> ) --> lLower
```

Arguments

<cString> Any character string

Returns

Description

This function takes the caracter string <cString> and checks to see if the leftmost character is a lowercased letter.If so, the function will return a logical true (.T.);otherwise, it will return a logical false (.F.).

Examples

```
? islower( 'ABCde' )      // .F.
? islower( 'aBCde' )      // .T.
```

Status

Ready

Compliance

This function is CA-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ISALPHA()
ISDIGIT()
ISUPPER()
LOWER()
UPPER()

LTRIM()

Removes leading spaces from a string

Syntax

LTRIM(<cString>) --> cReturn

Arguments

<cString> Character expression with leading spaces

Returns

LTRIM() returns a copy of the original string with leading spaces removed.

Description

This function trims the leading space blank

Examples

```
? LTRIM( "HELLO     " )
```

Status

Ready

Compliance

This functions is CA-CLIPPER compatible

Platforms

All

Files

Library is rtl

See Also:

TRIM()
RTRIM()
ALLTRIM()

AT()

Locates the position of a substring in a main string.

Syntax

AT(<cSearch>, <cString>, [<nStart>], [<nEnd>]) --> nPos

Arguments

<cSearch> Substring to search for

<cString> Main string

<nStart> First position to search in cString, by default 1

<nEnd> End posistion to search, by default cString length

Returns

AT() return the starting position of the first occurrence of the substring in the main string

Description

This function searches the string <cString> for the characters in the first string <cSearch>. If the substring is not contained within the second expression,the function will return 0. The third and fourth parameters lets you indicate a starting and end offset to search in.

Examples

```
QOUT( "at( 'cde', 'abcdefgfedcba' ) = '" +;
at( 'cde', 'abcsefgfedcba' ) + "'" )
```

Status

Ready

Compliance

This function is sensitive to HB_C52_STRICT settings during the compilation of source/rtl/at.c

<nStart> and <nEnd> are Harbour extensions and do not exist if HB_C52_STRICT is defined. In that case, the whole string is searched.

Platforms

All

Files

Library is rtl

See Also:

RAT()

RAT()

Searches for a substring from the right side of a string.

Syntax

RAT(<cSearch>, <cString>) --> nPos

Arguments

<cSearch>	Substring to search for
<cString>	Main string

Returns

RAT() return the location of beginnig position.

Description

This function searches througt <cString> for the first existence of <cSearch>.The search operation is performed from the right side of <cString> to the left. If the function is unable to find any occurence of <cSearch> in <cString>, the return value is 0.

Examples

```
QOUT( "rat( 'cde', 'abcdefgfedcba' ) = '" +;
rat( 'cde', 'abcsefgfedcba' ) + "'" )
```

Status

Ready

Compliance

Will not work with a search string > 64 KB on some platforms

Platforms

All

Files

Library is rtl

See Also:

AT()
SUBSTR()
RIGHT()

LEFT()

Extract the leftmost substring of a character expression

Syntax

LEFT(<cString>, <nLen>) --> cReturn

Arguments

<cString> Main character to be parsed

<nLen> Number of bytes to return beggining at the leftmost position

Returns

<cReturn> Substring of evaluation

Description

This functions returns the leftmost <nLen> characters of <cString>. It is equivalent to the following expression: <fixed> SUBSTR(<cString>, 1, <nLen>) </fixed>

This functions returns the leftmost <nLen> characters of <cString>. It is equivalent to

Examples

```
? LEFT( 'HELLO HARBOUR', 5 )    // HELLO
```

Status

Ready

Compliance

This functions is CA CLIPPER compatible

Platforms

All

Files

Library is rtl

See Also:

SUBSTR()
RIGHT()
AT()
RAT()

RIGHT()

Extract the rightmost substring of a character expression

Syntax

RIGHT(<cString>, <nLen>) --> cReturn

Arguments

<cString> Character expression to be parsed

<nLen> Number of bytes to return beggining at the rightmost position

Returns

<cReturn> Substring of evaluation

Description

This functions returns the rightmost <nLen> characters of <cString>. It is equivalent to the following expressions: <fixed> SUBSTR(<cString>, - <nLen>) SUBSTR(<cString>, LEN(<cString>) - <nLen> + 1, <nLen>) </fixed>

This functions returns the rightmost <nLen> characters of <cString>. It is equivalent to

Examples

```
? RIGHT( 'HELLO HARBOUR', 5 )     // RBOUR
```

Status

Ready

Compliance

This functions is CA CLIPPER compatible

Platforms

All

Files

Library is rtl

See Also:

SUBSTR()
LEFT()
AT()
RAT()

SUBSTR()

Returns a substring from a main string

Syntax

```
SUBSTR( <cString>, <nStart>, [<nLen>] ) --> cReturn
```

Arguments

<cString> Character expression to be parsed

<nStart> Start position

<nLen> Number of characters to return

Returns

<cReturn> Substring of evaluation

Description

This functions returns a character string formed from <cString>, starting at the position of <nStart> and continuing on for a lenght of <nLen> characters. If <nLen> is not specified, the value will be all remaining characters from the position of <nStart>.

The value of <nStart> may be negative. If it is, the direction of operation is reversed from a default of left-to-right to right-to-left for the number of characters specified in <nStart>. If the number of characters from <nStart> to the end of the string is less than <nLen> the rest are ignored.

Examples

```
? SUBSTR( 'HELLO HARBOUR' , 7, 4 )      // HARB
? SUBSTR( 'HELLO HARBOUR' ,-3, 3 )      // OUR
? SUBSTR( 'HELLO HARBOUR' , 7    )      // HARBOUR
```

Status

Ready

Compliance

This functions is CA-Clipper compatible with the execption that CA-Clipper will generate an error if the passed string is longer than 64Kb, and Harbour on some plataform is not limit by this size.

Platforms

All

Files

Library is rtl

See Also:

LEFT()
AT()
RIGHT()

STR()

Convert a numeric expression to a character string.

Syntax

STR(<nNumber>, [<nLength>], [<nDecimals>]) --> cNumber

Arguments

<nNumber> is the numeric expression to be converted to a character string.

<nLength> is the length of the character string to return, including decimal digits, decimal point, and sign.

<nDecimals> is the number of decimal places to return.

Returns

STR() returns <nNumber> formatted as a character string. If the optional length and decimal arguments are not specified, STR() returns the character string according to the following rules:

Expression	Return Value Length
Field Variable	Field length plus decimals
Expressions/constants	Minimum of 10 digits plus decimals
VAL()	Minimum of 3 digits
MONTH()/DAY()	3 digits
YEAR()	5 digits
RECNO()	7 digits

Description

STR() is a numeric conversion function that converts numeric values to character strings. It is commonly used to concatenate numeric values to character strings. STR() has applications displaying numbers, creating codes such as part numbers from numeric values, and creating index keys that combine numeric and character data.

STR() is like TRANSFORM(), which formats numeric values as character strings using a mask instead of length and decimal specifications.

The inverse of STR() is VAL(), which converts character numbers to numerics.

* If <nLength> is less than the number of whole number digits in <nNumber>, STR() returns asterisks instead of the number.

* If <nLength> is less than the number of decimal digits required for the decimal portion of the returned string, Harbour rounds the number to the available number of decimal places.

* If <nLength> is specified but <nDecimals> is omitted (no decimal places), the return value is rounded to an integer.

Examples

```
? STR( 10, 6, 2 ) // " 10.00"
? STR( -10, 8, 2 ) // "  -10.00"
```

Tests

see the regression test suit for comprehensive tests.

Status

Ready

Compliance

CA-Clipper compatible.

Files

Library is rtl

See Also:

STRZERO()
TRANSFORM()
VAL()

STRZERO()

Convert a numeric expression to a character string, zero padded.

Syntax

STRZERO(<nNumber>, [<nLength>], [<nDecimals>]) --> cNumber

Arguments

<nNumber> is the numeric expression to be converted to a character string.

<nLength> is the length of the character string to return, including decimal digits, decimal point, and sign.

<nDecimals> is the number of decimal places to return.

Returns

STRZERO() returns <nNumber> formatted as a character string. If the optional length and decimal arguments are not specified, STRZERO() returns the character string according to the following rules:

Expression	Return Value Length
Field Variable	Field length plus decimals
Expressions/constants	Minimum of 10 digits plus decimals
VAL()	Minimum of 3 digits
MONTH()/DAY()	3 digits
YEAR()	5 digits
RECNO()	7 digits

Description

STRZERO() is a numeric conversion function that converts numeric values to character strings. It is commonly used to concatenate numeric values to character strings. STRZERO() has applications displaying numbers, creating codes such as part numbers from numeric values, and creating index keys that combine numeric and character data.

STRZERO() is like TRANSFORM(), which formats numeric values as character strings using a mask instead of length and decimal specifications.

The inverse of STRZERO() is VAL(), which converts character numbers to numerics.

* If <nLength> is less than the number of whole number digits in <nNumber>, STR() returns asterisks instead of the number.

* If <nLength> is less than the number of decimal digits required for the decimal portion of the returned string, Harbour rounds the number to the available number of decimal places.

* If <nLength> is specified but <nDecimals> is omitted (no decimal places), the return value is rounded to an integer.

Examples

```
? STRZERO( 10, 6, 2 ) // "010.00"
? STRZERO( -10, 8, 2 ) // "-0010.00"
```

Tests

see the regression test suit for comprehensive tests.

Status

Ready

Compliance

CA-Clipper compatible (it was part of the samples).

Files

Library is rtl

See Also:

STR()

HB_VALTOSTR()

Converts any scalar type to a string.

Syntax

HB_VALTOSTR(<xValue>) --> cString

Arguments

<xValue> is any scalar argument.

Returns

<cString> A string representation of <xValue> using default conversions.

Description

HB_VALTOSTR can be used to convert any scalar value to a string.

Examples

```
? HB_VALTOSTR( 4 )
? HB_VALTOSTR( "String" )
```

Tests

```
? HB_VALTOSTR( 4 ) == "         4"
? HB_VALTOSTR( 4.0 / 2 ) == "         2.00"
? HB_VALTOSTR( "String" ) == "String"
? HB_VALTOSTR( CTOD( "01/01/2001" ) ) == "01/01/01"
? HB_VALTOSTR( NIL ) == "NIL"
? HB_VALTOSTR( .F. ) == ".F."
? HB_VALTOSTR( .T. ) == ".T."
```

Status

Ready

Compliance

HB_VALTOSTR() is a Harbour enhancement.

Files

Library is rtl

See Also:

STR()

LEN()

Returns size of a string or size of an array.

Syntax

```
LEN( <cString> | <aArray> ) --> <nLength>
```

Arguments

<acString> is a character string or the array to check.

Returns

Description

This function returns the string length or the size of an array. If it is used with a multidimensional array it returns the size of the first dimension.

Examples

```
? LEN( "Harbour" )           // 7
? LEN( { "One", "Two" } )    // 2
```

Tests

```
function Test()
   LOCAL cName := ""
   ACCEPT "Enter your name: " TO cName
   ? LEN( cName )
return nil
```

Status

Ready

Compliance

LEN() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

EMPTY()
RTRIM()
LTRIM()
AADD()
ASIZE()

EMPTY()

Checks if the passed argument is empty.

Syntax

```
EMPTY( <xExp> ) --> lIsEmpty
```

Arguments

<xExp> is any valid expression.

Returns

false (.F.).

Description

This function checks if an expression has empty value and returns a logical indicating whether it the expression is empty or not.

Examples

```
? EMPTY( "I'm not empty" )    // .F.
```

Tests

```
FUNCTION Test()
   ? EMPTY( NIL )             // .T.
   ? EMPTY( 0 )               // .T.
   ? EMPTY( .F. )             // .T.
   ? EMPTY( "" )              // .T.
   ? EMPTY( 1 )               // .F.
   ? EMPTY( .T. )             // .F.
   ? EMPTY( "smile" )         // .F.
   ? EMPTY( Date() )          // .F.
RETURN NIL
```

Status

Ready

Compliance

EMPTY() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

LEN()

DESCEND()

Inverts an expression of string, logical, date or numeric type.

Syntax

DESCEND(<xExp>) --> xExpInverted

Arguments

<xExp> is any valid expression.

Returns

Description

This function converts an expression in his inverted form. It is useful to build descending indexes.

Examples

```
// Seek for Smith in a descending index
SEEK DESCEND( "SMITH" )
```

Tests

```
DATA->( DBSEEK( DESCEND( "SMITH" ) ) )
will seek "SMITH" into a descending index.
```

Status

Ready

Compliance

DESCEND() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ARRAY()
ARRAY()

HB_ANSITOOEM()

Convert a windows Character to a Dos based character

Syntax

```
HB_ANSITOOEM( <cString> ) --> cDosString
```

Arguments

<cString> Windows ansi string to convert to DOS oem String

Returns

<cDosString> Dos based string

Description

This function converts each character in <cString> to the corresponding character in the MS-DOS (OEM) character set. The character expression <cString> should contain characters from the ANSI character set. If a character in <cString> doesn't have a MS-DOS equivalent, the character is converted to a similar MS-DOS character.

Examples

```
? HB_OEMTOANSI( "Harbour" )
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

This functions work only on Windows Plataform

Files

Library is rtl

See Also:

HB_OEMTOANSI()

HB_OEMTOANSI()

Convert a DOS(OEM) Character to a WINDOWS (ANSI) based character

Syntax

HB_OEMTOANSI(<cString>) --> cDosString

Arguments

<cString> DOS (OEM) string to convert to WINDOWS (ANSI) String

Returns

<cDosString> WINDOWS based string

Description

This function converts each character in <cString> to the corresponding character in the Windows (ANSI) character set. The character expression <cString> should contain characters from the OEM character set. If a character in <cString> doesn't have a ANSI equivalent, the character is remais the same.

Examples

```
? HB_OEMTOANSI( "Harbour" )
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

This functions work only on Windows Plataform

Files

Library is rtl

See Also:

HB_ANSITOOEM()

LOWER()

Universally lowercases a character string expression.

Syntax

LOWER(<cString>) --> cLowerString

Arguments

<cString> Any character expression.

Returns

<cLowerString> Lowercased value of <cString>

Description

This function converts any character expression passes as <cString> to its lowercased representation.Any nonalphabetic character withing <cString> will remain unchanged.

Examples

```
? LOWER( "HARBOUR" )      // harbour
? LOWER( "Hello All" )    // hello all
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Platforms

ALL

Files

Library is rtl

See Also:

UPPER()
ISLOWER()
ISUPPER()

UPPER()

Converts a character expression to uppercase format

Syntax

UPPER(<cString>) --> cUpperString

Arguments

<cString> Any character expression.

Returns

<cUpperString> Uppercased value of <cString>

Description

This function converts all alpha characters in <cString> to upper case values and returns that formatted character expression.

Examples

```
? UPPER( "harbour" )       // HARBOUR
? UPPER( "Harbour" )       // HARBOUR
```

Status

Ready

Compliance

This function is CA-Clipper compatible

Platforms

All

Files

Library is rtl

See Also:

LOWER()
ISUPPER()
ISLOWER()

CHR()

Converts an ASCII value to it character value

Syntax

```
CHR( <nAsciiNum> )  --> cReturn
```

Arguments

<nAsciiNum> Any ASCII character code.

Returns

<cReturn> Character expression of that ASCII value

Description

This function returns the ASCII character code for <nAsciiNum>.The number expressed must be an interger value within the range of 0 to 255 inclusive.The CHR() function will send the character returned to whatever device is presently set.

The CHR() function may be used for printing special codes as well as normal and graphics character codes.

Examples

```
? CHR( 32 )
? chr( 215 )
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ASC()
INKEY()

ASC()

Returns the ASCII value of a character

Syntax

```
ASC( <cCharacter> ) --> nAscNumber
```

Arguments

<cCharacter> Any character expression

Returns

<nAscNumber> ASCII value

Description

This function return the ASCII value of the leftmost character of any character expression passed as <cCharacter>.

Examples

```
? ASC( "A" )
? ASC( "¹" )
```

Status

Ready

Compliance

This function is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

CHR()

PADC()

Centers an expression for a given width

Syntax

```
PADC( <xVal>, <nWidth>, <cFill> )  --> cString
```

Arguments

<xVal> A Number, Character or Date value to pad

<nWidth> Width of output string

<cFill> Character to fill in the string

Returns

<cString> The Center string of <xVal>

Description

This function takes an date, number or character expression <xVal> and attempt to center the expression within a string of a given width expressed as <nWidth>. The default character used to pad either side of <xVal> will be a blank space. This character may be explicitly specified the value of <cFill>.

If the lenght of <xVal> is longer then <nWidth>,this function will truncate the string <xVal> from the leftmost side to the lenght of <nWidth>.

Examples

```
? PADC( 'Harbour',20 )
? PADC( 34.5142, 20 )
? PADC( Date(), 35 )
```

Tests

See Examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

ALLTRIM()
PADL()
PADR()

PADL()

Left-justifies an expression for a given width

Syntax

```
PADL( <xVal>, <nWidth>, <cFill> )  --> cString
```

Arguments

<xVal> An number,Character or date to pad

<nWidth> Width of output string

<cFill> Character to fill in the string

Returns

<cString> The left-justifies string of <xVal>

Description

This function takes an date,number,or character expression <xVal> and attempt to left-justify it within a string of a given width expressed as <nWidth>.The default character used to pad left side of <xVal> will be an blank space;however,this character may be explicitly specified the value of <cFill>.

If the lenght of <xVal> is longer then <nWidth>,this function will truncate the string <xVal> from the leftmost side to the lenght of <nWidth>.

Examples

```
? PADL( 'Harbour', 20 )
? PADL( 34.5142, 20 )
? PADL( Date(), 35 )
```

Tests

See examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

ALLTRIM()
PADC()
PADR()

PADR()

Right-justifies an expression for a given width

Syntax

PADR(<xVal>, <nWidth>, <cFill>) --> cString

Arguments

<xVal> A Number, Character or Date value to pad

<nWidth> Width of output string

<cFill> Character to fill in the string

Returns

<cString> The right-justifies string of <xVal>

Description

This function takes an date,number,or character expression <xVal> and attempt to right-justify it within a string of a given width expressed as <nWidth>.The default character used to pad right side of <xVal> will be an blank space;however,this character may be explicitly specified the value of <cFill>.

If the lenght of <xVal> is longer then <nWidth>,this function will truncate the string <xVal> from the leftmost side to the lenght of <nWidth>.

Examples

```
? PADR( 'Harbour', 20 )
? PADR( 34.5142, 20 )
? PADR( Date(), 35 )
```

Tests

See examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

ALLTRIM()
PADC()
PADL()

ALLTRIM()

Removes leading and trailing blank spaces from a string

Syntax

ALLTRIM(<cString>) --> cExpression

Arguments

<cString> Any character string

Returns

<cExpression> An string will all blank spaces removed from <cString>

Description

This function returns the string <cExpression> will all leading and trailing blank spaces removed.

Examples

```
? ALLTRIM( "HELLO HARBOUR" )
? ALLTRIM( "     HELLO HARBOUR" )
? ALLTRIM( "HELLO HARBOUR     " )
? ALLTRIM( "     HELLO HARBOUR     " )
```

Tests

See Examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

LTRIM()
RTRIM()
TRIM()

RTRIM()

Remove trailing spaces from a string.

Syntax

```
RTRIM( <cExpression> ) --> cString
```

Arguments

<cExpression> Any character expression

Returns

<cString> A formated string with out any blank spaced.

Description

This function returns the value of <cString> with any trailing blank removed.

This function is indentical to RTRIM() and the opposite of LTRIM(). Together with LTRIM(),this function equated to the ALLTRIM() function.

Examples

```
? RTRIM( "HELLO" )              //  "HELLO"
? RTRIM( "" )                   //  ""
? RTRIM( "UA   " )              //  "UA"
? RTRIM( "   UA" )              //  "   UA"
```

Tests

See Examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

ALLTRIM()
LTRIM()
TRIM()

TRIM()

Remove trailing spaces from a string.

Syntax

```
TRIM( <cExpression> )   --> cString
```

Arguments

<cExpression> Any character expression

Returns

<cString> A formated string with out any blank spaced.

Description

This function returns the value of <cString> with any trailing blank removed.

This function is indentical to RTRIM() and the opposite of LTRIM(). Together with LTRIM(),this function equated to the ALLTRIM() function.

Examples

```
? TRIM( "HELLO" )     //  "HELLO"
? TRIM( "" )          //   ""
? TRIM( "UA   " )     //   "UA"
? TRIM( "   UA" )     //   "   UA"
```

Tests

See Examples

Status

Ready

Compliance

This function is Ca-Clipper compilant

Platforms

All

Files

Library is rtl

See Also:

RTRIM()
LTRIM()
ALLTRIM()

REPLICATE()

Repeats a single character expression

Syntax

```
REPLICATE( <cString>, <nSize> )  --> cReplicateString
```

Arguments

<cString> Character string to be replicated

<nSize> Number of times to replicate <cString>

Returns

<cReplicateString> A character expression containg the <cString> fill character.

Description

This function returns a string composed of <nSize> repetitions of <cString>.The lenght of the character string returned by this function is limited to the memory avaliable.

A value of 0 for <nSize> will return a NULL string.

Examples

```
? REPLICATE( 'a', 10 )      // aaaaaaaaaa
? REPLICATE( 'b', 100000 )
```

Tests

See Examples

Status

Ready

Compliance

This function is Ca-Clipper compliant in all aspects, with the exception don't have the Clipper 64Kb string length.

Platforms

All

Files

Library is rtl

See Also:

SPACE()
PADC()
PADL()
PADR()

SPACE()

Returns a string of blank spaces

Syntax

SPACE(<nSize>) --> cString

Arguments

<nSize> The lenght of the string

Returns

<cString> A string containing blank spaces

Description

This function returns a string consisting of <nSize> blank spaces. If the value of <nSize> is 0, a NULL string ("") will be returned.

This function is useful to declare the lenght of a character memory variable.

Examples

```
FUNC MAIN
LOCAL cBigString
LOCAL cFirst
LOCAL cString := Space(20)   // Create an characte memory variable
                             // with lenght 20
? len(cString)      // 20
cBigString:=space(100000)    // create a memory variable with 100000
                             // blank spaces
?  len(cBigString)
Use Tests New
cFirst:= makeempty(1)
? len(cFirst)
Return Nil

Function MakeEmpty(xField)
LOCAL nRecord
LOCAL xRetValue

If !empty(alias())
    nRecord:=recno()
    dbgoto(0)
    if valtype(xField)=="C"
        xField:= ascan(dbstruct(),{|aFields| aFields[1]==upper(xfield)})
    else
        default xField to 0
        if xField < 1 .or. xField>fcount()
            xfield:=0
        endif
    endif
    if !(xfield ==0)
        xRetvalue:=fieldget(xfield)
    endif
    dbgoto(nrecord)
endif
return( xRetvalue)
```

Tests

See examples

Status

Ready

Compliance

This function is Ca-Clipper compliant in all aspects, with the exception don't have the Clipper 64Kb string length.

Platforms

All

Files

Library is rtl

See Also:

- PADC()
- PADL()
- PADR()
- REPLICATE()

VAL()

Convert a number from a character type to numeric

Syntax

```
VAL( <cNumber> ) --> nNumber
```

Arguments

<cNumber> Any valid character string of numbers.

Returns

<nNumber> The numeric value of <cNumber>

Description

This function converts any number previosly defined as an character expression <cNumber> into a numeric expression.

This functions is the oppose of the STR() function.

Examples

```
? VAL( '31421' ) // 31421
```

Tests

See regression test

Status

Ready

Compliance

This function is Ca-Clipper compatible

Platforms

All

Files

Library is rtl

See Also:

STR()
TRANSFORM()

STRTRAN()

Translate substring valuw with a main string

Syntax

```
STRTRAN( <cString>, <cLocString>, [<cRepString>], [<nPos>],
[<nOccurences>] ) --> cReturn
```

Arguments

<cString>	The main string to search
<cLocString>	The string to locate in the main string
<cRepString>	The string to replace the <cLocString>
<nPos>	The first occurence to be replaced
<nOccurences>	Number of occurence to replace

Returns

<cReturn>	Formated string

Description

This function searches for any occurence of <cLocString> in <cString> and replacesit with <cRepString>.If <cRepString> is not specified, a NULL byte will replace <cLocString>.

If <nPos> is used,its value defines the first occurence to be replaced.The default value is 1.Additionally,if used,the value of <nOccurences> tell the function how many occurrences of <cLocString> in <cString> are to the replaced.The default of <nOccurences> is all occurrences.

Examples

```
? STRTRAN( "Harbour  Power", "  ", " " )    // Harbour Power
         // Harbour Power The future  of  xBase
? STRTRAN( "Harbour  Power  The Future  of  xBase", "  ", " " ,, 2 )
```

Tests

See regression test

Status

Ready

Compliance

Will not work with a search string of > 64 KB on some platforms

Platforms

All

Files

Libraty is rtl

See Also:

SUBSTR()
AT()

TRANSFORM()

Formats a value based on a specific picture template.

Syntax

TRANSFORM(<xExpression>, <cTemplate>) --> cFormated

Arguments

<xExpression> Any expression to be formated.

<cTemplate> Character string with picture template

Returns

<cFormated> Formatted expression in character format

Description

This function returns <xExpression> in the format of the picture expression passed to the function as <cTemplate>.

Their are two components that can make up <cTemplate> : a function string and a template string.Function strings are those functions that globally tell what the format of <xExpression> should be.These functions are represented by a single character precede by the @ symbol.

There are a couple of rules to follow when using function strings and template strings:

- First, a single space must fall between the function template and the template string if they are used in conjunction with one another.

- Second,if both components make up the value of <cTemplate>,the function string must precede the template string.Otherwise,the function string may appear with out the template string and vice versa.

The table below shows the possible function strings avaliable with the TRANSFORM() function.

@B	Left justify the string within the format.
@C	Issue a CR after format is numbers are positive.
@D	Put dates in SET DATE format.
@E	Put dates in BRITISH format.
@L	Make a zero padded string out of the number.
@R	Insert nontemplate characters.
@X	Issue a DB after format is numbers are negative.
@Z	Display any zero as blank spaces.
@(	Quotes around negative numbers
@!	Convert alpha characters to uppercased format.

The second part of <cTemplate> consists of the format string.Each character in the string may be formated based on using the follow characters as template markers for the string.

A,N,X,9,#	Any data type
L	Shows logical as "T" or "F"
Y	Shows logical as "" or "N"
!	Convert to uppercase
$	Dolar sing in place of leading spaces in numeric expression
*	Asterisks in place of leading spaces in numeric expression
,	Commas position
.	Decimal point position

Examples

```
LOCAL cString := 'This is harbour'
LOCAL nNumber := 9923.34
LOCAL nNumber1 := -95842.00
LOCAL lValue := .T.
LOCAL dDate := DATE()
? 'working with String'
? "Current String is", cString
? "All uppercased", TRANSFORM( cString, "@!" )
? "Date is", ddate
? "Date is ", TRANSFORM( ddate, "@D" )
? TRANSFORM( nNumber, "@L 99999999" )  //  "009923.34"
? TRANSFORM( 0       , "@L 9999"     )  //  "0000"
```

Tests

See regression Test

Status

Ready

Compliance

The @L function template is a FOXPRO/Xbase Extension

Platforms

All

Files

Library is rtl

See Also:

@...SAY
DEVOUTPICT()

Strong Typing
Compile-Time type checking

Description

Strong Type Checking, could also be described as "Compile-Time Type Checking". As you might know Clipper, generates a Run-Time Error, ("Type Mismatch") when we attempt to perform some operations with the wrong type of Variable.

Examples:

```
LOCAL Var1 := "A"

? Var1 * 3 // Error here.

@ Var1, 7 SAY 'Hello' // Error here.

? SubStr( "Hello", Var1 ) // Error here.
```

The above 3 lines would all result in Run-Time Error, because Var1 is of type CHARACTER but the above lines used it as if it was of type NUMERIC.

Using Strong Type Checking, or Compile-Time Type Checking, the above problem would have been discovered and reported in COMPILE-TIME, rather than waiting for the inevitable problem to be discovered when we finally execute the program.

Strong Typed Languages allow the programmer to "tell" the compiler (declare) what is the type of a each Variable, so that the Compiler in return can warn the programmer, when ever such Declared (Strong Typed) Variable, is used in a context which is incompatible with its declared type.

For instance, if we "told" the compiler that Var1 above is of type CHARACTER (LOCAL Var1 AS CHARACTER) the Harbour Compiler could, in return, warn us if we attempted to perform the calculation:

```
Var1 * 3
```

because the Compiler knows we can't perform a multiplication of a Character. (we might allow it in some context, but this is beyond the scope of this discussion). Similarly we would have been warned when attempting to use Var1 as a Row Number (@ Var1), or as the 2nd operand of the SubStr() function SubStr("Hello", Var1)), because the Compiler knows that these operations require a NUMERIC rather than CHARACTER type.

The above may save us lots of time, by pointing a problem, we can not escape, since such code will never perform correctly once executed. So rather than wait to the testing cycle, for such problems to be discovered, (and some times even later, after we may have distributed our applications) instead we may know of such problems as soon as we type HARBOUR ProgName -w3

Harbour also offers a hybrid mode, where it can report such type mismatch problems, even without requiring the programmer to declare the type of variables. This feature, is referred to as Adaptive Type Checking. The programmer, is not required to make any changes in his code, to take advantage of this feature. All of the above 3 errors would have been reported just as effectively as if the programmer Strong Typed (declared) Var1. Harbour would have been able to report such problems at compile time,because the assignment Var1 := "A" implied that Var1 is of type CHARACTER,until it will be assigned another value. Therefore Harbour will "remember" that Var1 "adapted" type CHARACTER, and thus the subsequent multiplication Var1 * 3, will be reported as an error, as soon as you attempt to compile such code.

The nice aspect of this hybrid mode, is that unlike Strong Typed Variables,you don't have to declare the type, so no code changes are need, the Type instead is assumed by implication (type of the assigned value). The other benefit, is that it is completely ok to assign a new value of different type, any time, to such undeclared (variant) variable. As soon as we assign a new type, the Compiler will than protect us from using the Variable in an incompatible context, since the variable "adapted" this type as soon as we assigned a value which implies a type.

While Adapted Type Checking may be fairly effective in reporting many common mistakes, to take full benefits of such Compile-Time checking, it is recommended to do declare the Type of Variables, when ever possible.

The Harbour Strong Type features, also allows the declaration of the expected parameters (including optionals) of User Defined Functions, as well as their return Type. Similarly, you may declare the Type of any Class Variables, Methods, and Methods Parameters.

BASE/1003

Attempt to acces nonexisting or hidden variable

Description

The specified variable was not found.
If it is a database field make sure that the required database is open.
If it is a private or public variable then you must first create it using PRIVATE or PUBLIC statement.

Functions

Status

Clipper

BASE/1068

Invalid type of argument

Description

The used data is not of logical type

Functions

Status

Clipper

BASE/1068

Bound error in array access

Description

The attempt to retrieve data from non-array value

Functions

Status

Clipper

BASE/1069

Bound error in array access

Description

The attempt to set data to non-array value

Functions

Status

Clipper

BASE/1078
Invalid type of arguments

Description

The type of compared arguments do not match

Functions

==

Status

Clipper

BASE/1072

Invalid type of arguments

Description

The type of compared arguments do not match

Functions

<>

Status

Clipper

BASE/1073
Invalid type of arguments

Description

The type of compared argument do not match

Functions

<

Status

Clipper

BASE/1074
Invalid type of arguments

Description

The type of compared arguments do not match

Functions

<=

Status

Clipper

BASE/1075

Invalid type of arguments

Description

The type of compared arguments do not match

Functions

>

Status

Clipper

BASE/1076
Invalid type of arguments

Description

The type of compared arguments do not match

Functions

>=

Status

Clipper

BASE/1077

Invalid type of arguments

Description

Operation is not allowed for passed argument. The argument is not a logical value.

Functions

!

Status

Clipper

BASE/1078

Invalid type of arguments

Description

The type of one or both arguments is not a logical

Functions

.AND.

Status

Clipper

BASE/1079
Invalid type of arguments

Description

The type of one or both arguments is not a logical

Functions

.OR.

Status

Clipper

BASE/1076

Invalid type of arguments

Description

The value of argument cannot be incremented

Functions

++

Status

Clipper

BASE/1081
Invalid type of arguments

Description

The plus operation is not allowed for used arguments.

Functions

+

Status

Clipper

BASE/1082
Invalid type of arguments

Description

The minus operation is not allowed for used arguments.

Functions

-

Status

Clipper

BASE/1100
Incorrect type of argument

Description

The specified argument is not a string.

Functions

RTRIM, TRIM

Status

Clipper

BASE/1101

Incorrect type of argument

Description

The specified argument is not a string.

Functions

LTRIM

Status

Clipper

BASE/1102

Invalid argument passed to function

Description

The first argument passed to a function is not a string.

Functions

UPPER

Status

Clipper

BASE/1103
Invalid argument passed to function

Description

The first argument passed to a function is not a string.

Functions

LOWER

Status

Clipper

BASE/1104

Incorrect type of argument

Description

The specified argument is not a numeric value.

Functions

CHR

Status

Clipper

BASE/1105
Invalid argument passed to function

Description

The arguments passed to a function are of incorrect type.

Functions

SPACE

Status

Clipper

BASE/1106

Invalid argument passed to function

Description

The arguments passed to a function are of incorrect type.

Functions

REPLICATE

Status

Clipper

BASE/1107

Incorrect type of argument

Description

The specified argument is not a string.

Functions

ASC

Status

Clipper

BASE/1108
Incorrect type of argument

Description

The specified argument is not a string.

Functions

AT

Status

Clipper

BASE/1076
Invalid type of arguments

Status

Clipper

BASE/1110

Invalid argument passed to function

Description

The first argument passed to a function is not a string.

Functions

SUBSTR

Status

Clipper

BASE/1110
Invalid argument passed to function

Description

The passed argument is neither a string nor an array.

Functions

LEN

Status

Clipper

BASE/1112

Invalid argument passed to function

Description

The argument (or arguments) passed to a function are of incorrect type

Functions

YEAR

Status

Clipper

BASE/1113
Invalid argument passed to function

Description

The argument (or arguments) passed to a function are of incorrect type

Functions

MONTH

Status

Clipper

BASE/1114

Invalid argument passed to function

Description

The argument (or arguments) passed to a function are of incorrect type

Functions

DAY

Status

Clipper

BASE/1115

Invalid argument passed to function

Description

The argument (or arguments) passed to a function are of incorrect type

Functions

DOW

Status

Clipper

BASE/1116
Invalid argument passed to function

Description

The argument (or arguments) passed to a function are of incorrect type

Functions

CMONTH

Status

Clipper

BASE/1117
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is of incorrect type

Functions

CDOW

Status

Clipper

BASE/1120
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is of incorrect type

Functions

DTOS

Status

Clipper

BASE/1122

Incorrect type of argument

Description

The argument (or arguments) passed to a function is of incorrect type

Functions

TRANSFORM

Status

Clipper

BASE/1124

Incorrect type of argument

Description

The first argument is not a string.

Functions

LEFT

Status

Clipper

BASE/1126

Invalid argument passed to function

Description

The first arguments passed to a function is not a string.

Functions

STRTRAN

Status

Clipper

BASE/1132

Bound error in array access

Description

The specified index into an array was greater then the number of elements in the array.

Functions

Status

Clipper

BASE/1133

Bound error in array assigment

Description

The specified index into an array was greater then the number of elements in the array.

Functions

Status

Clipper

BASE/1068

Bound error in array element assigment

Description

The specified index into an array was greater then the number of elements in the array.

Functions

Status

Clipper

BASE/1085

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

MOD

Status

Clipper

BASE/1089

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

ABS

Status

Clipper

BASE/1090
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

INT

Status

Clipper

BASE/1092

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

MIN

Status

Clipper

BASE/1093

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

MAX

Status

Clipper

BASE/1094

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

ROUND

Status

Clipper

BASE/1095
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

LOG

Status

Clipper

BASE/1096
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

EXP

Status

Clipper

BASE/1097
Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not an numeric value

Functions

SQRT

Status

Clipper

BASE/1098

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not a string value

Functions

VAL

Status

Clipper

BASE/1099

Invalid argument passed to function

Description

The argument (or arguments) passed to a function is not a numeric value

Functions

STR

Status

Clipper

BASE/2010
Incorrect arguments type

Description

Passed Run Time Errors was not strings with filenames to copy/

Functions

__COPYFILE

Compliance

Harbour specific

BASE/2012

File error

Description

An error has occured during the attempt to open, create or write during copy operation

Functions

__COPYFILE

Status

Clipper

BASE/2017
Invalid argument passed to a function

Description

The first argument is not an array or/and the second argument is not a code block

Functions

AEVAL

Status

Clipper

BASE/2020
Invalid argument passed to function

Description

The passed value is negative. Only values > 0 are allowed.

Functions

SET DECIMALS
SET EPOCH
SET MARGIN
SET MESSAGE

Status

Clipper

BASE/3001

Incorrect argument type

Description

The passed argument is not an object. Only data of type OBJECT can be cloned by this function

Functions

OCLONE

Status

Harbour specific

BASE/3002

Super class does not return an object

Description

Passed argument is not a name of defined class or specified class doesn't have a super class

Functions

__INSTSUPER

Status

Harbour specific

BASE/3003

Cannot find super class

Description

Passed argument is not a name of defined class

Functions

__INSTSUPER

Status

Harbour specific

BASE/3004

Cannot modify a DATA item in a class

Description

The attempt to modify a data member of a class was made. Only INLINE and METHOD can be modified

Functions

CLASSMOD

Status

Harbour specific

BASE/3005

Incorrect arguments type

Description

Either the first argument was not an object or the second argument wasn't a string.

Functions

ISMESSAGE, OSEND

Status

Harbour specific

BASE/3007
Invalid type of argument

Description

The passed arguments are causing conflict in hanndling of the request. There is no point in waiting forever for no input events!

Functions

INKEY

Status

Harbour specific

BASE/3008
Invalid type of argument

Description

The passed argument(s) is not a string. It should be a string with a variable name or an one-dimensional array of strings.

Functions

__MVPRIVATE, __MVPUBLIC

Status

Harbour specific

BASE/3009

Incorrect argument passed to __MVGET function

Description

__MVGET function expects only one argument: a string with a name of variable. The value of this variable will be returned.

Functions

__MVGET

Status

Harbour specific

BASE/3010

Incorrect argument passed to __MVPUT function

Description

__MVPUT function expects at least one argument: a string with a name of variable. The value of this variable will be set.

Functions

__MVPUT

Status

Harbour specific

BASE/3011

Invalid argument passed to a function

Description

The attempt to retrieve the function argument that was not passed. The number of requested argument is greated then the number of passed arguments.

Functions

PVALUE

Status

Harbour specific

BASE/3012

Invalid argument passed to a function

Description

The first argument is not a string with function/procedure name that should be called.

Functions

DO

Status

Harbour specific

BASE/3101

Invalid argument passed to an object/class function

Description

One passed argument is not of the required type.

Functions

__OBJ*()

Status

Harbour specific

BASE/3102

A symbol should be modified or deleted from a class, but the symbol

Description

A symbol should be modified or deleted from a class, but the symbol doesn't exist.

Functions

__OBJ*()

Status

Harbour specific

BASE/3103

A symbol should be added to a class, but the symbol already exists.

Description

A symbol should be added to a class, but the symbol already exists.

Functions

__OBJ*()

Status

Harbour specific

TOOLS/4001

Invalid argument passed to function

Description

The second arguments passed to a function is not a string.

Functions

ISLEAPYEAR

Status

Harbour specific

TERM/2013

Create error

Description

The specified file cannot be created due some OS error.

Functions

SET, SET ALTERNATE TO

Status

Clipper

TBROWSENew()
Create a Browse Object

Constructor syntax

TBROWSENew(<nTop>,<nLeft>,<nBottom>,<nRight>)   --> <oBrowse>

Arguments

<nTop>	Top Row
<nLeft>	Top Left Column
<nBottom>	Bottom Row
<nRight>	Bottom Right Column

Returns

<oBrowse> An new Browse Object

Description

This function set up a browsing window at top-left coordinates of <nTop>,<nLeft> to bottom-right coordinates of <nBottom>,<nRight>. To browse Database files use TBROWSEDB() function insted.

Data

:aColumns	Array to hold all browse columns
:autoLite	Logical value to control highlighting
:cargo	User-definable variable
:colorSpec	Color table for the TBrowse display
:colPos	Current cursor column position
:colSep	Column separator character
:footSep	Footing separator character
:freeze	Number of columns to freeze
:goBottomBlock	Code block executed by TBrowse:goBottom()
:goTopBlock	Code block executed by TBrowse:goTop()
:headSep	Heading separator character
:hitBottom	Indicates the end of available data
:hitTop	Indicates the beginning of available data
:leftVisible	Indicates position of leftmost unfrozen column in display
:nBottom	Bottom row number for the TBrowse display
:nLeft	Leftmost column for the TBrowse display
:nRight	Rightmost column for the TBrowse display
:nTop	Top row number for the TBrowse display
:rightVisible	Indicates position of rightmost unfrozen column in display
:rowCount	Number of visible data rows in the TBrowse display
:rowPos	Current cursor row position
:skipBlock	Code block used to reposition data source
:stable	Indicates if the TBrowse object is stable

:aRedraw	Array of logical items indicating, is appropriate row need to be redraw
:RelativePos	Indicates record position relatively position of first record on the screen
:lHeaders	Internal variable which indicates whether there are column footers to paint
:lFooters	Internal variable which indicates whether there are column footers to paint
:aRect	The rectangle specified with ColorRect()
:aRectColor	The color positions to use in the rectangle specified with ColorRect()
:aKeys	Holds the Default movement keys

Method

New(nTop, nLeft, nBottom, nRight) Create an new Browse class and set the default values

Down()	Moves the cursor down one row
End()	Moves the cursor to the rightmost visible data column
GoBottom()	Repositions the data source to the bottom of file
GoTop()	Repositions the data source to the top of file
Home()	Moves the cursor to the leftmost visible data column
Left()	Moves the cursor left one column
PageDown()	Repositions the data source downward
PageUp()	Repositions the data source upward
PanEnd()	Moves the cursor to the rightmost data column
PanHome()	Moves the cursor to the leftmost visible data column
PanLeft()	Pans left without changing the cursor position
PanRight()	Pans right without changing the cursor position
Right()	Moves the cursor right one column
Up()	Moves the cursor up one row
ColCount()	Return the Current number of Columns
ColorRect()	Alters the color of a rectangular group of cells
ColWidth(nColumn)	Returns the display width of a particular column
Configure(nMode)	Reconfigures the internal settings of the TBrowse object nMode is an undocumented parameter in CA-Cl*pper
LeftDetermine()	Determine leftmost unfrozen column in display
DeHilite()	Dehighlights the current cell
DelColumn(nPos)	Delete a column object from a browse
ForceStable()	Performs a full stabilization
GetColumn(nColumn)	Gets a specific TBColumn object
Hilite()	Highlights the current cell
InsColumn(nPos, oCol)	Insert a column object in a browse
Invalidate()	Forces entire redraw during next stabilization

RefreshAll()	Causes all data to be recalculated during the next stabilize
RefreshCurrent()	Causes the current row to be refilled and repainted on next stabilize
SetColumn(nColumn, oCol)	Replaces one TBColumn object with another
Stabilize()	Performs incremental stabilization
DispCell(nColumn, cColor)	Displays a single cell

Examples

See tests/testbrw.prg

Tests

See tests/testbrw.prg

Status

Started

Compliance

This functions is Compatible with Ca-Clipper 5.2. The applykey() and Setkey() methods are only visible if HB_COMPAT_C53 is defined.

Platforms

All

Files

Library is rtl

See Also:

TBROWSENew()
ARRAY()

SetKey()

Get an optionaly Set an new Code block associated to a inkey value

Syntax

SetKey(<nKey>[,<bBlock>]) --> bOldBlock

Arguments

<nKey> An valid inkey Code

<bBlock> An optional action to associate to the inkey value.

Returns

<bOldBlock> If an Keypress has it code block changes, it will return the previus one; otherwise, it will return the current one

Description

This method Get an optionaly set an code block that is associated to an inkey value. The table below show the default keypress/Code Block definitions

Inkey Value	Code Block
K_DOWN	{|Ob,nKey| Ob:Down(),0}
K_END	{|Ob,nKey| Ob:End(),0}
K_CTRL_PGDN	{|Ob,nKey| Ob:GoBottom(),0}
K_CTRL_PGUP	{|Ob,nKey| Ob:GoTop(),0}
K_HOME	{|Ob,nKey| Ob:Home(),0}
K_LEFT	{|Ob,nKey| Ob:Left(),0}
K_PGDN	{|Ob,nKey| Ob:PageDown(),0}
K_PGUP	{|Ob,nKey| Ob:PageUp(),0}
K_CTRL_END	{|Ob,nKey| Ob:PanEnd(),0}
K_CTRL_HOME	{|Ob,nKey| Ob:PanHome(),0}
K_CTRL_LEFT	{|Ob,nKey| Ob:PanLeft(),0}
K_CTRL_RIGHT	{|Ob,nKey| Ob:PanRight(),0}
K_RIGHT	{|Ob,nKey| Ob:Right(),0}
K_UP	{|Ob,nKey| Ob:Up(),0}
K_ESC	{|Ob,nKey| -1 }

The keys handlers can be queried,added and replace an removed from the internal keyboard dictionary. See the example.

oTb:SETKEY(K_TAB,{|oTb,nKey| -1})

An default key handler can be declared by specifyin a value of 0 for <nKey>.It associate code block will be evaluated each time TBrowse:Applykey() is called with an key value that is not contained in the dictionary. For example

oTb:SetKey(0,{|oTb,nKey| DefKeyHandler(otb,nkey}) This call the a function named DefKeyHandler() when nKey is not contained in the dictionary.

To remove an keypress/code block definition, specify NIL for <bBlock>
oTb:SetKey(K_ESC,nil)

Examples

```
oTb:SeyKey(K_F10,{|otb,nkey| ShowListByname(otb)}
```

Applykey()

Evaluates an code block associated with an specific key

Syntax

ApplyKey(<nKey>) --> nResult

Arguments

<nKey> An valid Inkey code

Returns

<nResult> Value returned from the evaluated Code Block See Table Below

Value	Meaning
-1	User request for the browse lost input focus
0	Code block associated with <nkey> was evaluated
1	Unable to locate <nKey> in the dictionary,Key was not processed

Description

This method evaluate an code block associated with <nkey> that is contained in the TBrowse:setkey() dictionary.

Examples

```
while .t.
   oTb:forceStable()
   if (oTb:applykey(inkey(0))==-1)
      exit
   endif
enddo
```

AddColumn()

Add an New Column to an TBrowse Object

Syntax

AddColumn(oCol) --> Self

Arguments

<oCol>	Is an TbColumn object

Returns

<Self>	The Current object

Description

This method add an new column object specified as <oCol> to the assigned browsing object.

HBClass()

HBClass() is used in the creation of all classes

Syntax

```
oClass := HBClass():New("TMyClass")
-or-
HBClass() is usually accessed by defining a class with the commands
defined in hbclass.h:
CLASS HBGetList// Calls HBClass() to create the HBGetList class
...
ENDCLASS
```

Arguments

Returns

create the classes you define.

Description

HBClass is a class that ... The class methods are as follows:

New() Create a new instance of the class

Examples

```
FUNCTION TestObject()
local oObject

oObject := HBClass():New("TMyClass")
oObject:End()

RETURN Nil
```

Status

Ready

Compliance

Object Oriented syntax in Harbour is compatible with CA-CLIPPER. But Clipper only allowed creation of objects from a few standard classes, and did not let the programmer create new classes. In Harbour, you can create your own classes--complete with Methods, Instance Variables, Class Variables and Inheritance. Entire applications can be designed and coded in Object Oriented style.

Platforms

All

Files

Library is rtl

See Also:

__objHasData()
ARRAY()
CLASS

__XSaveScreen()

Save whole screen image and coordinate to an internal buffer

Syntax

```
__XSaveScreen() --> NIL
```

Arguments

Returns

__XSaveScreen() always return NIL.

Description

__XSaveScreen() save the image of the whole screen into an internal buffer, it also save current cursor position. The information could later be restored by __XRestScreen(). Each call to __XSaveScreen() overwrite the internal buffer.

SAVE SCREEN command is preprocessed into __XSaveScreen() function during compile time. Note that SAVE SCREEN TO is preprocessed into SAVESCREEN() function.

__XSaveScreen() is a compatibility function, it is superseded by SAVESCREEN() which allow you to save part or all the screen into a variable.

Examples

```
// save the screen, display list of files than restore the screen
SAVE SCREEN
DIR *.*
WAIT
RESTORE SCREEN
```

Status

Ready

Compliance

__XSaveScreen() works exactly like CA-Clipper's __XSaveScreen()

Platforms

__XSaveScreen() is part of the GT API, and supported only by some platforms.

Files

Library is rtl

See Also:

RESTORE SCREEN
ARRAY()
ARRAY()

SAVE SCREEN

Save whole screen image and coordinate to an internal buffer

Syntax

```
SAVE SCREEN
```

Arguments

Returns

Description

SAVE SCREEN save the image of the whole screen into an internal buffer, it also save current cursor position. The information could later be restored by REST SCREEN. Each call to SAVE SCREEN overwrite the internal buffer.

SAVE SCREEN command is preprocessed into __XSaveScreen() function during compile time. Note that SAVE SCREEN TO is preprocessed into SAVESCREEN() function.

Examples

```
// save the screen, display list of files than restore the screen
SAVE SCREEN
DIR *.*
WAIT
RESTORE SCREEN
```

Status

Ready

Compliance

__XSaveScreen() works exactly like CA-Clipper's __XSaveScreen()

Platforms

__XSaveScreen() is part of the GT API, and supported only by some platforms.

See Also:

RESTORE SCREEN
__XRestScreen()
__XSaveScreen()

__XRestScreen()

Restore screen image and coordinate from an internal buffer

Syntax

```
__XRestScreen() --> NIL
```

Arguments

Returns

__XRestScreen() always return NIL.

Description

__XRestScreen() restore saved image of the whole screen from an internal buffer that was saved by __XSaveScreen(), it also restore cursor position. After a call to __XRestScreen() the internal buffer is cleared.

RESTORE SCREEN command is preprocessed into __XRestScreen() function during compile time. Note that RESTORE SCREEN FROM is preprocessed into RESTSCREEN() function.

__XRestScreen() is a compatibility function, it is superseded by RESTSCREEN() which allow you to restore the screen from a variable.

Examples

```
// save the screen, display list of files than restore the screen
SAVE SCREEN
DIR *.*
WAIT
RESTORE SCREEN
```

Status

Ready

Compliance

__XRestScreen() works exactly like CA-Clipper's __XRestScreen()

Platforms

__XRestScreen() is part of the GT API, and supported only by some platforms.

Files

Library is rtl

See Also:

__XRestScreen()
SAVE SCREEN
__XSaveScreen()

RESTORE SCREEN

Restore screen image and coordinate from an internal buffer

Syntax

RESTORE SCREEN

Arguments

Returns

Description

Rest Screen restore saved image of the whole screen from an internal buffer that was saved by Save Screen, it also restore cursor position. After a call to Rest Screen the internal buffer is cleared.

RESTORE SCREEN command is preprocessed into __XRestScreen() function during compile time. Note that RESTORE SCREEN FROM is preprocessed into RESTSCREEN() function.

Examples

```
// save the screen, display list of files than restore the screen
SAVE SCREEN
DIR *.*
WAIT
RESTORE SCREEN
```

Status

Ready

Compliance

Rest Screen() works exactly like CA-Clipper's Rest Screen

Platforms

Rest Screen is part of the GT API, and supported only by some platforms.

See Also:

__XRestScreen()
SAVE SCREEN
__XSaveScreen()

ALERT()

Display a dialog box with a message

Syntax

ALERT(<xMessage>, [<aOptions>], [<cColorNorm>], [<nDelay>]) --> nChoice or NIL

Arguments

<xMessage> Message to display in the dialog box. can be of any Harbour type. If <xMessage> is an array of Character strings, each element would be displayed in a new line. If <xMessage> is a Character string, you could split the message to several lines by placing a semicolon (;) in the desired places.

<aOptions> Array with available response. Each element should be Character string. If omitted, default is { "Ok" }.

<cColorNorm> Color string to paint the dialog box with. If omitted, default color is "W+/R".

<nDelay> Number of seconds to wait to user response before abort. Default value is 0, that wait forever.

Returns

ALERT() return Numeric value representing option number chosen. If ESC was pressed, return value is zero. The return value is NIL if ALERT() is called with no parameters, or if <xMessage> type is not Character and HB_C52_STRICT option was used. If <nDelay> seconds had passed without user response, the return value is 1.

Description

ALERT() display simple dialog box on screen and let the user select one option. The user can move the highlight bar using arrow keys or TAB key. To select an option the user can press ENTER, SPACE or the first letter of the option.

If the program is executed with the //NOALERT command line switch, nothing is displayed and it simply returns NIL. This switch could be overridden with __NONOALERT().

If the GT system is linked in, ALERT() display the message using the full screen I/O system, if not, the information is printed to the standard output using OUTSTD().

Examples

```
LOCAL cMessage, aOptions, nChoice

// harmless message
cMessage := "Major Database Corruption Detected!;" +  ;
            "(deadline in few hours);;"             +  ;
            "where DO you want to go today?"

// define response option
aOptions := { "Ok", "www.jobs.com", "Oops" }

// show message and let end user select panic level
nChoice := ALERT( cMessage, aOptions )
DO CASE
   CASE nChoice == 0
        // do nothing, blame it on some one else
   CASE nChoice == 1
        ? "Please call home and tell them you're gonn'a be late"
   CASE nChoice == 2
        // make sure your resume is up to date
   CASE nChoice == 3
        ? "Oops mode is not working in this version"
ENDCASE
```

Status

Ready

Compliance

This function is sensitive to HB_C52_STRICT settings during the compilation of source/rtl/alert.prg

defined: <xMessage> accept Character values only and return NIL if other types are passed.

undefined: <xMessage> could be any type, and internally converted to Character string. If type is Array, multi-line message is displayed.

defined: Only the first four valid <aOptions> are taken.

undefined: <aOptions> could contain as many as needed options.

If HB_COMPAT_C53 was define during compilation of source/rtl/alert.prg the Left-Mouse button could be used to select an option.

The interpretation of the //NOALERT command line switch is done only if HB_C52_UNDOC was define during compilation of source/rtl/alert.prg

<cColorNorm> is a Harbour extension, or at least un-documented in Clipper 5.2 NG.

<nDelay> is a Harbour extension.

Files

Library is rtl

See Also:

@...PROMPT
MENU TO
OUTSTD()
__NONOALERT()

__NONOALERT()

Override //NOALERT command line switch

Syntax

__NONOALERT() --> NIL

Arguments

Returns

__NONOALERT() always return NIL.

Description

The //NOALERT command line switch cause Clipper to ignore calls to the ALERT() function, this function override this behavior and always display ALERT() dialog box.

Examples

```
// make sure alert are been displayed
__NONOALERT()
```

Status

Ready

Files

Library is rtl

Compliance

__NONOALERT() is an undocumented CA-Clipper function and exist only if HB_C52_UNDOC was defined during the compilation of source/rtl/alert.prg

HB_OSNEWLINE()

Returns the newline character(s) to use with the current OS

Syntax

HB_OSNewLine() --> cString

Returns

<cString> A character string containing the character or characters required to move the screen cursor or print head to the start of a new line. The string will hold either CHR(10) or CHR(13) + CHR(10).

Description

Returns a character string containing the character or characters required to move the screen cursor or print head to the start of a new line for the operating system that the program is running on (or thinks it is running on, if an OS emulator is being used).

Examples

```
// Get the newline character(s) for the current OS using defaults.
STATIC s_cNewLine
...
s_cNewLine := HB_OSNewLine()
...
OutStd( "Hello World!" + s_cNewLine )
...
```

Tests

```
valtype( HB_OSNewLine() ) == "C"
LEN( HB_OSNewLine() ) == 1
```

Status

Ready

Compliance

This is an add-on Operating System Tool function.

Platforms

Under OS_UNIX_COMPATIBLE operating system the return value is the Line-Feed (0x0a) character CHR(10), with other operating systems (like DOS) the return value is the Carriage-Return plus Line-Feed (0x0d 0x0a) characters CHR(13)+CHR(10).

Files

Library is rtl

See Also:

OS()
OUTSTD()
OUTERR()

hb_ColorIndex()

Extract one color from a full Clipper colorspec string.

Syntax

hb_ColorIndex(<cColorSpec>, <nIndex>)

Arguments

<cColorSpec> is a Clipper color list

<nIndex> is the position of the color item to be extracted, the first position is the zero.

Returns

Description

Clipper has a color spec string, which has more than one color in it, separated with commas. This function is able to extract a given item from this list. You may use the manifest constants defined in color.ch to extract common Clipper colors.

Examples

```
? hb_ColorIndex( "W/N, N/W", CLR_ENHANCED ) // "N/W"
```

Tests

see the regression test suit for comprehensive tests.

Status

Ready

Compliance

Was not part of CA-Clipper.

Files

Library is rtl

See Also:

ARRAY()

DEVOUTPICT()

Displays a value to a device using a picture template

Syntax

DEVOUTPICT(<xExp>, <cPicture>, [<cColorString>]) --> NIL

Arguments

<xExp> is any valid expression.

<cPicture> is any picture transformation that TRANSFORM() can use.

<cColorString> is an optional string that specifies a screen color to use in place of the default color when the output goes to the screen.

Returns

Description

Outputs any expression using a picture transformation instead of using the default transformation for the type of expression.

Examples

```
// Output a negative dollar amount using debit notation.
DEVOUTPICT( -1.25, "@D$ 99,999.99 )
```

Tests

```
@ 3,1 SAY -1.25 PICTURE "@D$ 99,999.99"
will display "$(     1.25)" starting on row four, column two of the
current device (without the double quotation marks, of course).
```

Status

Ready

Compliance

DEVOUTPICT() is mostly CA-Clipper compliant. Any differences are due to enhancements in the Harbour TRANSFORM() over CA-Clipper.

Files

Library is rtl

See Also:

ARRAY()
TRANSFORM()

__INPUT()

Stops application

Syntax

__INPUT(<cMessage>) --> <cString>

Arguments

<cMessage> is any valid expression.

Returns

Description

This function waits for a console input and returns macroed expression entered.

Status

Started

Compliance

__INPUT() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

__WAIT()
ARRAY()

__TextSave()

Redirect console output to printer or file and save old settings

Syntax

```
__TextSave( <cFile> ) --> NIL
```

Arguments

<cFile> is either "PRINTER" (note the uppercase) in which console output is SET to PRINTER, or a name of a text file with a default ".txt" extension, that is used to redirect console output.

Returns

__TextSave() always return NIL.

Description

__TextSave() is used in the preprocessing of the TEXT TO command to redirect the console output while saving old settings that can be restored later by __TextRestore().

Status

Ready

Compliance

__TextSave() is an Undocumented CA-Clipper function

Platforms

ALL

Files

Library is rtl

See Also:

SET()
SET ALTERNATE
SET PRINTER
ARRAY()
__TextRestore()

__TextRestore()

Restore console output settings as saved by __TextSave()

Syntax

__TextRestore() --> NIL

Arguments

Returns

__TextRestore() always return NIL.

Description

__TextRestore() is used in the preprocessing of the TEXT TO command to restore console output settings that were previously saved by __TextSave().

Status

Ready

Compliance

__TextRestore() is an Undocumented CA-Clipper function

Platforms

All

Files

Library is rtl

See Also:

SET()
SET ALTERNATE
SET PRINTER
ARRAY()
__TextSave()

__WAIT()

Stops the application until a key is pressed.

Syntax

```
__WAIT( <cMessage> ) --> <cKey>
```

Arguments

<cMessage> is a string.

Returns

Description

This function stops the application until a key is pressed. The key must be in the range 32..255. Control keys are not processed.

Examples

```
// Wait for a key stroke
__Wait( "Press a key to continue" )
```

Tests

```
do while cKey != "Q"
  cKey := __Wait( "Press 'Q' to continue" )
end do
```

Status

Ready

Compliance

__WAIT() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

ARRAY()
__INPUT()

OUTSTD()

Write a list of values to the standard output device

Syntax

```
OUTSTD( <xExp,...> ) --> NIL
```

Arguments

<xExp,...> is a list of expressions to display. Expressions are any mixture of Harbour data types.

Returns

OUTSTD() always returns NIL.

Description

OUTSTD() write one or more values into the standard output device. Character and Memo values are printed as is, Dates are printed according to the SET DATE FORMAT, Numeric values are converted to strings, Logical values are printed as .T. or .F., NIL are printed as NIL, values of any other kind are printed as empty string. There is one space separating each two values. Note that Numeric value can take varying length when converted into string depending on its source (see STR() for detail).

OUTSTD() is similar to QQOUT() with the different that QQOUT() send its output to the Harbour console stream, which can or can not be redirected according with the screen driver, and OUTSTD() send its output to the standard output device (STDOUT) and can be redirected.

Examples

```
OUTSTD( "Hello" )            // Result: Hello

OUTSTD( 1, .T., NIL, "A" )
OUTSTD( "B" )                // Result:          1 .T. NIL AB
```

Status

Ready

Compliance

OUTSTD() works exactly as in CA-Clipper

Files

Library is rtl

See Also:

- ARRAY()
- ARRAY()
- ARRAY()
- DEVOUTPICT()
- ARRAY()
- ARRAY()
- OUTERR()
- ARRAY()
- ARRAY()
- STR()

OUTERR()

Write a list of values to the standard error device

Syntax

```
OUTERR( <xExp,...> ) --> NIL
```

Arguments

<xExp,...> is a list of expressions to display. Expressions are any mixture of Harbour data types.

Returns

OUTERR() always returns NIL.

Description

OUTERR() write one or more values into the standard error device. Character and Memo values are printed as is, Dates are printed according to the SET DATE FORMAT, Numeric values are converted to strings, Logical values are printed as .T. or .F., NIL are printed as NIL, values of any other kind are printed as empty string. There is one space separating each two values. Note that Numeric value can take varying length when converted into string depending on its source (see STR() for detail).

There is an undocumented CA-Clipper command line switch //STDERR which can set the file handle to write output from OUTERR(). If not specified the default STDERR is used, //STDERR or //STDERR:0 set OUTERR() to output to the same file handle as OUTSTD(), //STDERR:n set output to file handle n. Like other undocumented features this switch is available only if source/rtl/console.c was compiled with the HB_C52_UNDOC flag.

Examples

```
// write error log information
OUTERR( DATE(), TIME(), "Core meltdown detected" )
```

Status

Ready

Compliance

OUTERR() works exactly as in CA-Clipper

Files

Library is rtl

See Also:

ARRAY()
ARRAY()
ARRAY()
DEVOUTPICT()
ARRAY()
ARRAY()
OUTSTD()
ARRAY()
ARRAY()
STR()

EJECT

Issue an command to advance the printer to the top of the form

Syntax

```
EJECT
```

Arguments

Description

This command issue an form-feed command to the printer.If the printer is not properly hooked up to the computer,an error will not be generated and the command will be ignored.

Once completed,the values of PROW() and PCOL(),the row and column indicators to the printer,will be set to 0.Their values,however,may be manipulated before or after ussuing an EJECT by using the DEVPOS() function.

On compile time this command is translated into __EJECT() function.

Examples

```
Use Clientes New
Set Device to Printer
CurPos:=0
While !Eof()
? Clientes->nome,Clientes->endereco
Curpos++
if Curpos >59
    Curpos:=0
    Eject
Endif
Enddo
Set Device to Screen
Use
```

Tests

See examples

Status

Ready

Compliance

This command is Ca-Clipper compliant

Platforms

All

See Also:

ARRAY()
SET PRINTER
ARRAY()
ARRAY()

COL()

Returns the current screen column position

Syntax

```
COL()  --> nPosition
```

Arguments

Returns

<nPosition>   Current column position

Description

This function returns the current cursor column position.The value for this function can range between 0 and MAXCOL().

Examples

```
? Col()
```

Status

Ready

Compliance

This Functions is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

ROW()
MAXROW()
MAXCOL()

ROW()

Returns the current screen row position

Syntax

```
ROW()  --> nPosition
```

Arguments

Returns

<nPosition>   Current screen row position

Description

This function returns the current cursor row location.The value for this function can range between 0 and MAXCOL().

Examples

```
? Row()
```

Status

Ready

Compliance

This Functions is Ca-Clipper compliant

Platforms

All

Files

Library is rtl

See Also:

COL()
MAXROW()
MAXCOL()

MAXCOL()

Returns the maximun number of columns in the current video mode

Syntax

```
MAXCOL()  --> nPosition
```

Arguments

Returns

<nPosition>   The maximun number of columns possible in current video mode

Description

This function returns the current cursor column position.The value for this function can range between 0 and MAXCOL().

Examples

```
? MAXCol()
```

Status

Ready

Compliance

This Functions is Ca-Clipper compliant.

Platforms

It works in all platform with some remarks:Under Linux and OS/2 the number of columns avaliable depends of the current Terminal screen size.Under Win32, the return value of MAXCOL() function is only affected if called after an SETMODE() function

Files

Library is rtl

See Also:

ROW()
MAXROW()
COL()

MAXROW()

Returns the current screen row position

Syntax

```
MAXROW()  --> nPosition
```

Arguments

Returns

<nPosition>   The maximun number of rows possible in current video mode

Description

This function returns the current cursor row location.The value for this function can range between 0 and MAXCOL().

Examples

```
? MAXROW()
```

Status

Ready

Compliance

This Functions is Ca-Clipper compliant

Platforms

It works in all platform with some remarks:Under Linux and OS/2 the number of columns avaliable depends of the current Terminal screen size.Under Win32, the return value of MAXROW() function is only affected if called after an SETMODE() function

Files

Library is rtl

See Also:

COL()
ROW()
MAXCOL()

READVAR()

Return variable name of current GET or MENU

Syntax

READVAR([<cVarName>]) --> cOldVarName

Arguments

<cVarName> is a new variable name to set.

Returns

READVAR() return the old variable name. If no variable previously was set, READVAR() return "".

Description

READVAR() is set inside a READ or MENU TO command to hold the uppercase name of the GET / MENU TO variable, and re-set back to old value when those commands finished. You should not normally set a variable name but rather use it to retrieve the name of a GET variable when executing a VALID or WHEN clause, or during SET KEY execution and you are inside a READ or MENU TO.

Examples

```
// display a menu, press F1 to view the MENU TO variable name
CLS
@ 1, 10 PROMPT "blood sucking insect that infect beds   "
@ 2, 10 PROMPT "germ; virus infection                   "
@ 3, 10 PROMPT "defect; snag; (source of) malfunctioning"
@ 4, 10 PROMPT "small hidden microphone                 "
@ 6, 10 SAY "(Press F1 for a hint)"
SET KEY 28 TO ShowVar
MENU TO What_Is_Bug

PROCEDURE ShowVar
ALERT( READVAR() )        // WHAT_IS_BUG in red ALERT() box
```

Status

Ready

Compliance

READVAR() works exactly like CA-Clipper's READKEY(), note however, that the <cVarName> parameter is not documented and used internally by CA-Clipper.

Platforms

All

Files

Library is rtl

See Also:

@...Get
@...PROMPT
MENU TO
ARRAY()
SET KEY
__AtPrompt()
__MenuTo()

LABEL FORM

Displays labels to the screen or an alternate device

Syntax

```
LABEL FORM <cLabelName> [TO PRINTER] [TO FILE <cFile>] [<cScope>]
[WHILE <bWhile> ] [FOR <bFor> ] [SAMPLE] [NOCONSOLE]
```

Arguments

<cLabelName>	Name of label file
<cFile>	Name of an alternate file
<cScope>	Expression of a scoping condition
<bWhile>	WHILE condition
<bFor>	FOR condition

Description

This command allows labels to be printed based on the format outlined in LBL file specified as <cLabelName>. By default, output will go to the screen however this output may be rerouted with either the TO PRINTER or the TO FILE clause.

If the TO FILE clause is specified, the name of the ASCII text file containing the generated labels will be <cFile>.

If no file extension is specified a .TXT extension is added. <cScope> is the scope condition for this command. Valid scopes include NEXT <expN> (number of records to be displayed, where <expN> is the number of records), RECORD <expN> (a specific record to be printed), REST (all records starting from the current record position,and ALL (all records). The default is ALL.

Both logical expression may work ill conjunction with one another where <bFor> is the logical expression for the FOR condition (for records to be displayed whitin a given value range) and <bWhile> for the WHILE condition (for records to be displayed until they fail to meet the condition).

If the SAMPLE clause is specified, test labels will be generated.

If the NOCONSOLE clause is specified,the console will be turned off while this command is being executed.

This command follows the search criteria outlined in the SET PATH TO command. The path may be specified, along, with (the drive letter, in <cLabelName>

Examples

```
FUNCTION MAIN()
USE Test New
LABEL FORM EE
USE
RETURN NIL
```

Status

Ready

Compliance

This command is CA-Clipper compliant.

Platforms

ALL

Files

Library is rtl

See Also:

REPORT FORM

REPORT FORM

Display a report

Syntax

```
REPORT FORM <cReportName> [TO PRINTER] [TO FILE <cFile>] [<cScope>]
[WHILE <bWhile> ] [FOR <bFor> ]
[PLAIN |HEADING <cHeading>] [NOEJECT] [SUMMARY]
[NOCONSOLE]
```

Arguments

<cReportName>	Name of report file
<cFile>	Name of alternate file
<cScope>	Scope.
<bWhile>	Logical expression of WHILE condition .
<bFor>	Logical expression of FOR condition.
<cHeading>	Report heading

Returns

Description

This command prints out the report named <cReportName>, which is a standard FRM file. The file extension is not required because FRM will be assumed. The SET PATH TO and SET DEFAULT TO commands affect the search for the file <cReportName>; unless a drive and path are specified in <cReportName>, REPORT will search the path specified in the SET PATH command if it cannot find the report form in the current directory.

The output of the report will be offset based on the setting of the SET MARGIN TO value.

By default, output will go to the console; however, it may be controlled via either the TO PRINTER or TO FILE clause. If the output is to go to the file, the name of the alternate file is specified in <cFile>. Unless specified in <cFile>, the default file extension will be TXT.

<cScope> is the scope for this command. Valid scopes include NEXT <expN> (where <expN> is the number of records), RECORD <expN> (a specific record to be displayed), REST (all records from the current record position), and ALL (all records). The default is ALL.

Both logical expressions may work in conjuntion with one another, where <bFor> is the logical expression for the FOR condition (for records to be displayed within a given range) and <bWhile> for the WHILE condition (for records to be displayed until the condition fails).

If the PLAIN clause is specified, date and page numbers are suppressed. In addition, there is no automatic page breaking, and the report title and column headings appear only once at the top of the form.

If the HEADING clause is used, <cHeading> is displayed on the first title of each report page. The value of <cHeading> is evaluated only once before executing the report; varying the values of <cHeading> is not allowed. The PLAIN clause will take precedence over the HEADING clause if both are included.

If the NOEJECT clause is used, the initial page eject on the report will not be issued when the output clause TO PRINTER is specified. Otherwise, this clause has no effect.

If the SUMMARY Clause is specified, the report will contain only groups, subgroups, and grand total information. The detailed title item information will be ignored.

If the NOCONSOLE clause is specified,output to the console will be turned off while this command is being executed.

Examples

```
FUNCTION() MAIN
USE Test New
Report FORM EE
USE
RETURN NIL
```

Status

Ready

Compliance

This Command is CA-Clipper compliant.

Platforms

ALL

Files

Library is rtl

See Also:

LABEL FORM

__MVPUBLIC()

This function creates a PUBLIC variable

Syntax

__MVPUBLIC(<variable_name>)

Arguments

<variable_name> = either a string that contains the variable's name or an one-dimensional array of strings with variable names No skeleton are allowed here.

Returns

Description

This function can be called either by the harbour compiler or by user. The compiler always passes the item of IT_SYMBOL type that stores the name of variable. If a variable with the same name exists already then the new variable is not created - the previous value remains unchanged. If it is first variable with this name then the variable is initialized with .T. value.

Examples

None Avaliable

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

__MVPRIVATE()

This function creates a PRIVATE variable

Syntax

__MVPRIVATE(<variable_name>)

Arguments

<variable_name> = either a string that contains the variable's name or an one-dimensional array of strings with variable names No skeleton are allowed here.

Returns

Description

This function can be called either by the harbour compiler or by user. The compiler always passes the item of IT_SYMBOL type that stores the name of variable. If a variable with the same name exists already then the value of old variable is hidden until the new variable is released. The new variable is always initialized to NIL value.

Examples

None Avaliable

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

__MVXRELEASE()

This function releases value stored in PRIVATE or PUBLIC variable

Syntax

__MVXRELEASE(<variable_name>)

Arguments

<variable_name> = either a string that contains the variable's name or an one-dimensional array of strings with variable names No skeleton are allowed here.

Returns

Description

This function releases values stored in memory variable. It shouldn't be called directly, rather it should be placed into RELEASE command. If the released variable is a PRIVATE variable then previously hidden variable with the same name becomes visible after exit from the procedure where released variable was created. If you access the released variable in the same function/procedure where it was created the the NIL value is returned. You can however assign a new value to released variable without any side effects.

It releases variable even if this variable was created in different procedure

Examples

```
PROCEDURE MAIN()
PRIVATE mPrivate

  mPrivate :="PRIVATE from MAIN()"
  ? mPrivate     //PRIVATE from MAIN()
  Test()
  ? mPrivate     //PRIVATE from MAIN()

RETURN

PROCEDURE Test()
PRIVATE mPrivate

  mPrivate :="PRIVATE from Test()"
  ? mPrivate           //PRIVATE from TEST()
  RELEASE mPrivate
  ? mPrivate           //NIL
  mPrivate :="Again in Test()"

RETURN
```

Status

Ready
This function is a Harbour extension

Files

Library is vm

__MVRELEASE()

This function releases PRIVATE variables

Syntax

__MVRELEASE(<skeleton>, <include_exclude_flag>)

Arguments

<skeleton> = string that contains the wildcard mask for variables' names that will be released. Supported wildcards: '*' and '?' <include_exclude_flag> = logical value that specifies if variables that match passed skeleton should be either included in deletion (if .T.) or excluded from deletion (if .F.)

Returns

Description

This function releases values stored in memory variables. It shouldn't be called directly, it should be placed into RELEASE ALL command. If the released variable is a PRIVATE variable then previously hidden variable with the same name becomes visible after exit from the procedure where released variable was created. If you access the released variable in the same function/procedure where it was created the the NIL value is returned. You can however assign a new value to released variable without any side effects. PUBLIC variables are not changed by this function.

Examples

None Avaliable

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

__MVSCOPE()

If variable exists then returns its scope.

Syntax

__MVSCOPE(<cVarName>)

Arguments

<cVarName> = a string with a variable name to check

Returns

=variable is not declared (not found in symbol table) HB_MV_UNKNOWN =if variable doesn't exist (but found in symbol table) HB_MV_ERROR =if information cannot be obtained (memory error or argument error) HB_MV_PUBLIC =for public variables HB_MV_PRIVATE_GLOBAL =for private variables declared outside of current function/procedure HB_MV_PRIVATE_LOCAL =for private variables declared in current function/procedure

Examples

```
PROCEDURE MAIN()
PUBLIC mPublic
PRIVATE mPrivateGlobal

CallProc()
? __mvScope( "mPrivateLocal" )      //HB_MV_UNKNOWN

RETURN

PROCEDURE CallProc()
PRIVATE mPrivateLocal

? __mvScope( "mPublic" )            //HB_MV_PUBLIC
? __mvScope( "mPrivateGlobal" )     //HB_MV_PRIVATE_GLOBAL
? __mvScope( "mPrivateLocal" )      //HB_MV_PRIVATE_LOCAL
? __mvScope( "mFindMe" )            //HB_MV_NOT_FOUND

IF( __mvScope( "mPublic" ) > HB_MV_ERROR )
   ? "Variable exists"
ELSE
   ? "Variable not created yet"
ENDIF

RETURN
```

Status

Ready
This function is a Harbour Extension

Files

Library is vm

See Also:

ARRAY()

__MVCLEAR()

This function releases all PRIVATE and PUBLIC variables

Syntax

```
__MVCLEAR()
```

Arguments

Returns

Description

This function releases all PRIVATE and PUBLIC variables. It is used to implement CLEAR MEMORY statement. The memory occupied by all visible variables are released - any attempt to access the variable will result in a runtime error. You have to reuse PRIVATE or PUBLIC statement to create again the variable that was cleared by this function.

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

See Also:

__MVPUBLIC()

__MVDBGINFO()

This function returns the information about the variables for debugger

Syntax

__MVDBGINFO(<nScope> [, <nPosition> [, @<cVarName>]])

Arguments

<nScope> = the scope of variables for which an information is asked Supported values (defined in hbmemvar.ch) HB_MV_PUBLIC HB_MV_PRIVATE (or any other value) <nPosition> = the position of asked variable on the list of variables with specified scope - it should start from position 1 <cVarName> = the value is filled with a variable name if passed by reference and <nPosition> is specified

Returns

Description

This function retrieves the information about memvar variables. It returns either the number of variables with given scope (when the first argument is passed only) or a value of variable identified by its position in the variables' list (when second argument is passed). It also returns the name of a variable if optional third argument is passed by reference.

If requested variable doesn't exist (requested position is greater then the number of defined variables) then NIL value is returned and variable name is set to "?"

The dynamic symbols table is used to find a PUBLIC variable then the PUBLIC variables are always sorted alphabetically. The PRIVATE variables are sorted in the creation order.

Note: Due to dynamic nature of memvar variables there is no guarantee that successive calls to retrieve the value of <Nth> PUBLIC variable will return the value of the same variable.

Examples

```
#include <hbmemvar.ch>

LOCAL nCount, i, xValue, cName

nCount =_mvDBGINFO( HB_MV_PUBLIC )
FOR i:=1 TO nCount
    xValue =__mvDBGINFO( HB_MV_PUBLIC, i, @cName )
    ? i, cName, xValue
NEXT

#include <hbmemvar.ch>
PROCEDURE MAIN()

? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )

PUBLIC cPublic:='cPublic in MAIN'

? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )

PRIVATE cPrivate:='cPrivate in MAIN'

? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )

CountMemvars()

? 'Back in Main'
? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )


RETURN
```

```
PROCEDURE CountMemvars()
LOCAL i, nCnt, xVal, cName
PUBLIC ccPublic:='ccPublic'
PRIVATE ccPrivate:='ccPrivate'

? 'In CountMemvars'
? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )

PRIVATE cPublic:='cPublic'

? 'PUBLIC=', __mvDBGINFO( HB_MV_PUBLIC )
? 'PRIVATE=', __mvDBGINFO( HB_MV_PRIVATE )

nCnt =__mvDBGINFO( HB_MV_PRIVATE ) +1
FOR i:=1 TO nCnt
  xVal =__mvDBGINFO( HB_MV_PRIVATE, i, @cName )
  ? i, '=', cName, xVal
NEXT

nCnt =__mvDBGINFO( HB_MV_PUBLIC ) +1
FOR i:=1 TO nCnt
  xVal =__mvDBGINFO( HB_MV_PUBLIC, i, @cName )
  ? i, '=', cName, xVal
NEXT

RETURN
```

Status

Ready

Compliance

This function should be called from the debugger only.

Files

Library is vm

__MVEXIST()

Determine if a given name is a PUBLIC or PRIVATE memory variable

Syntax

__MVEXIST(<cVarName>) --> <lVariableExist>

Arguments

<cVarName> - string that specifies the name of variable to check

Returns

__MVEXIST() return TRUE (.T.) if a MEMVAR named <cVarName> exist.

Description

This function determine if a PUBLIC or PRIVATE variable with the name <cVarName> exist or not.

Examples

```
LOCAL   TheLocal
STATIC  TheStatic
PUBLIC  ThePublic
PRIVATE ThePrivate
? __MVEXIST( "NotExist"   )        // .F.
? __MVEXIST( "TheLocal"   )        // .F.
? __MVEXIST( "TheStatic"  )        // .F.
? __MVEXIST( "ThePublic"  )        // .T.
? __MVEXIST( "ThePrivate" )        // .T.
```

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

See Also:

MEMVAR
ARRAY()
ARRAY()

__MVGET()

This function returns value of memory variable

Syntax

```
__MVGET( <cVarName> )  --> <xVar>
```

Arguments

<cVarName> - string that specifies the name of variable

Returns

<xVar> The value of variable

Description

This function returns the value of PRIVATE or PUBLIC variable if this variable exists otherwise it generates a runtime error. The variable is specified by its name passed as the function parameter.

Examples

```
FUNCTION MEMVARBLOCK( cMemvar )
RETURN {|x| IIF( PCOUNT()==0, __MVGET( cMemvar ),;
__MVPUT( cMemvar, x ) ) }
```

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

See Also:

__MVPUT()

__MVPUT()

This function set the value of memory variable

Syntax

__MVGET(<cVarName> [, <xValue>]) --> <xValue>

Arguments

<cVarName> - string that specifies the name of variable <xValue> - a value of any type that will be set - if it is not specified then NIL is assumed

Returns

<xValue> A value assigned to the given variable.

Description

This function sets the value of PRIVATE or PUBLIC variable if this variable exists otherwise it generates a runtime error. The variable is specified by its name passed as the function parameter. If a value is not specified then the NIL is assumed

Examples

```
FUNCTION MEMVARBLOCK( cMemvar )
RETURN {|x| IIF( PCOUNT()==0, __MVGET( cMemvar ),;
__MVPUT( cMemvar, x ) ) }
```

Status

Ready

Compliance

This function is a Harbour extension

Files

Library is vm

See Also:

__MVPUT()

MEMVARBLOCK()

Returns a codeblock that sets/gets a value of memvar variable

Syntax

MEMVARBLOCK(<cMemvarName>) --> <bBlock>

Arguments

<cMemvarName> - a string that contains the name of variable

Returns

<bBlock> a codeblock that sets/get the value of variable

Description

This function returns a codeblock that sets/gets the value of PRIVATE or PUBLIC variable. When this codeblock is evaluated without any parameters passed then it returns the current value of given variable. If the second parameter is passed for the codeblock evaluation then its value is used to set the new value of given variable - the passed value is also returned as a value of the codeblock evaluation.

Examples

```
PROCEDURE MAIN()
LOCAL cbSetGet
PUBLIC xPublic

cbSetGet = MEMVARBLOCK( "xPublic" )
EVAL( cbSetGet, "new value" )
? "Value of xPublic variable", EVAL( cbSetGet )

RETURN
```

Status

Ready

Compliance

This function is Ca-Clipper compatible

Files

Library is rtl

See Also:

MVGET()
MVPUT()

FIELDBLOCK()

Return a code block that sets/gets a value for a given field

Syntax

FIELDBLOCK(<cFieldName>) --> bFieldBlock

Arguments

<cFieldName> is a string that contain the field name.

Returns

FIELDBLOCK() return a code block that when evaluate could retrieve field value or assigning a new value to the field. If <cFieldName> is not specified or from type other than character, FIELDBLOCK() return NIL.

Description

FIELDBLOCK() return a code block that sets/gets the value of field. When this code block is evaluated without any parameters passed then it returns the current value of the given field. If the code block is evaluated with a parameter, than its value is used to set a new value to the field, this value is also return by the block. If the block is evaluate and there is no field with the name <cFieldName> in the current work area, the code block return NIL.

Note that FIELDBLOCK() works on the current work area, if you need a specific work area code block use FIELDWBLOCK() instead.

Examples

```
// open a file named Test that have a field named "name"
LOCAL bField
bFiled := FIELDBLOCK( "name" )
USE Test
? 'Original value of field "name" :', EVAL( bField )
EVAL( bField, "Mr X new name" )
? 'New value for the field "name" :', EVAL( bField )
```

Status

Ready

Compliance

If the block is evaluate and there is no field with the name <cFieldName> in the current work area, the code block return NIL.

CA-Clipper would raise BASE/1003 error if the field does not exist.

Files

Library is rtl

See Also:

EVAL()
FIELDWBLOCK()
MEMVARBLOCK()

FIELDWBLOCK()

Return a sets/gets code block for field in a given work area

Syntax

FIELDWBLOCK(<cFieldName>, <nWorkArea>) --> bFieldBlock

Arguments

<cFieldName> is a string that contain the field name.

<nWorkArea> is the work area number in which <cFieldName> exist.

Returns

FIELDWBLOCK() return a code block that when evaluate could retrieve field value or assigning a new value for a field in a given work area. If <cFieldName> is not specified or from type other than character, or if <nWorkArea> is not specified or is not numeric FIELDWBLOCK() return NIL.

Description

FIELDWBLOCK() return a code block that sets/gets the value of field from a given work area. When this code block is evaluated without any parameters passed then it returns the current value of the given field. If the code block is evaluated with a parameter, than its value is used to set a new value to the field, this value is also return by the block. If the block is evaluate and there is no field with the name <cFieldName> in work area number <nWorkArea>, the code block return NIL.

Examples

```
LOCAL bField
// this block work on the field "name" that exist on work area 2
bFiled := FIELDBLOCK( "name", 2 )
// open a file named One in work area 1
// that have a field named "name"
SELECT 1
USE One
// open a file named Two in work area 2
// it also have a field named "name"
SELECT 2
USE Two
SELECT 1
? "Original names: ", One->name, Two->name
? "Name value for file Two :", EVAL( bField )
EVAL( bField, "Two has new name" )
? "and now: ", One->name, Two->name
```

Status

Ready

Compliance

If the block is evaluate and there is no field with the name <cFieldName> in the given work area, the code block return NIL.

CA-Clipper would raise BASE/1003 error if the field does not exist.

Files

Library is rtl

See Also:

EVAL()
FIELDBLOCK()
MEMVARBLOCK()

TYPE()

Retrieves the type of an expression

Syntax

TYPE(<cExp>) --> <cRetType>

Arguments

<cExp> must be a character expression.

Returns

<cRetType> a string indicating the type of the passed expression.

<cRetType>	Meaning
"A"	array
"B"	block
"C"	string
"D"	date
"L"	logical
"M"	memo
"N"	numeric
"O"	object
"U"	NIL, local, or static variable, or not linked-in function
"UE"	syntax error in the expression or invalid arguments
"UI"	function with non-reserved name was requested

Description

This function returns a string which represents the data type of the argument. The argument can be any valid Harbour expression. If there is a syntax error in passed expression then "UE" is returned. If there is a call for any non-reserved Harbour function then "UI" is returned (in other words there is no call for passed UDF function during a data type determination - this is Clipper compatible behavior). Additionally if requested user defined function is not linked into executable then "U" is returned.

The data type of expression is checked by invoking a macro compiler and by evaluation of generated code (if there is no syntax errors). This causes that TYPE() cannot determine a type of local or static variables - only symbols visible at runtime can be checked.

Notice the subtle difference between TYPE and VALTYPE functions. VALTYPE() function doesn't call a macro compiler - it simply checks the type of passed argument of any type. TYPE() requires a string argument with a valid Harbour expression - the data type of this expression is returned.

Examples

```
? TYPE( "{ 1, 2 }" )                              //prints "A"
? TYPE( "IIF(.T., SUBSTR('TYPE',2,1), .F.)" )   //prints "C"
? TYPE( "AT( 'OK', MyUDF())>0" )                 //prints "UI"
? TYPE( "{ 1, 2 }[ 5 ]" )                        //prints "UE"

//--------------------------------------------------------

LOCAL c
PRIVATE a:="A", b:="B"
? TYPE( "a + b + c" )     //prints: "U" ('C' variable is a local one)

//--------------------------------------------------------

LOCAL cFilter := SPACE( 60 )
```

```
ACCEPT "Enter filter expression:" TO cFilter
IF( TYPE( cFilter ) $ "CDLMN" ) )
   // this is a valid expression
   SET FILTER TO &cFilter
ENDIF
```

Status

Ready

Compliance

- Incompatibility with Clipper: In the following code:

```
PRIVATE lCond := 0  ? TYPE( "IIF( lCond, 'true', MyUDF() )" )
```

Clipper will print "UE" - in Harbour the output will be "UI"

- if "UI" is returned then the syntax of the expression is correct. However invalid arguments can be passed to function/procedure that will cause runtime errors during evaluation of expression.

Files

Library is rtl

See Also:

VALTYPE()

VALTYPE()

Retrieves the data type of an expression

Syntax

```
VALTYPE( <xExp> ) --> <cReturnType>
```

Arguments

<xExp> is any valid expression.

Returns

<cReturnType> a character indicating the type of the passed expression.

Description

This function returns one character which represents the date type of the argument.

Examples

See Test

Tests

```
function Test()
   ? ValType( Array( 1 ) )  --> "A"
   ? ValType( {|| 1 + 1 } ) --> "B"
   ? ValType( "HARBOUR" )   --> "C"
   ? ValType( Date() )      --> "D"
   ? ValType( .T. )         --> "L"
   ? ValType( 1 )           --> "N"
   ? ValType( TBrowse() )   --> "O"
   ? ValType( NIL )         --> "U"
return nil
```

Status

Ready

Compliance

VALTYPE() is fully CA-Clipper compliant.

Files

Library is rtl

See Also:

TYPE()

Macro compiler

Description

```
Invoking the macro compiler:
==============================

&variable
or
&( expression )
or
&variable.text
```

HB_SETMACRO()

Enable/disable the macro compiler runtime features.

Syntax

HB_SETMACRO(<nOption>, [<lOnOff>]) --> <lOldSetting>

Arguments

<nOption> One of the HB_SM_* constants defined in set.ch.

<lOnOff> .T. to enable or .F. to disable a feature

Returns

HB_SETMACRO() return the old state of requested feature.

Description

This function enables or disables some features of the macro compiler. The Harbour is extending the macro features compared to an original set available in Clipper. Enabling/disabling some of them allows to keep strict Clipper compatibility.

Available features are: HB_SM_HARBOUR - enables harbour extensions: operators: ++, --, +=, -=, *=, /=, ^= objects: assigments to an instance variable **HB_SM_XBASE - enables other xbase dialects extensions: expanding of expresions lists HB_SM_SHORTCUTS - enables optimized evaluation of logical operators (.and., .or.) HB_SM_PREPROC - enables preprocessing of commands** This is meaningfull if Harbour is compiled with HB_MACRO_STATEMENTS flag

```
INIT PROCEDURE IWANTCLIPPER()
  HB_SETMACRO( HB_SM_HARBOUR, .F. )
  HB_SETMACRO( HB_SM_XBASE, .F. )
RETURN
```

Status

Ready

Compliance

This function is Harbour extension.

Platforms

All

Files

Header file is set.ch Library is macro

See Also:

Macro compiler

__dbDelim()

Copies the contents of a database to a delimited text file or

Syntax

```
__dbDelim( <lExport>, <xcFile>, [<xcDelim>], [<aFields>],
[<bFor>], [<bWhile>], [<nNext>], [<nRecord>], <lRest>  ) --> NIL
```

Arguments

<lExport> If set to .T., copies records to a delimited file. If set to .F., append records from a delimited file. <xcFile> The name of the text file to copy to or append from. If a file extension is not specified, ".txt" is used by default. <xcDelim> Either the character to use as the character field delimiter (only the first character is used). or "BLANK" (not case sensitive), which eliminates the character field delimiters and sets the field separator to a single space instead of a comma. <aFields> An aray of field names to limit the processint to. If not specified, or if empty, then all fields are processed. <bFor> An optional code block containing a FOR expression that will reduce the number of records to be processed. <bWhile> An optional code block containing a WHILE expression that will reduce the number of records to be processed. <nNext> If present, but nRecord is not present, specifies to process this number of records, starting with the current record. A value of 0 means to process no records. <nRecord> If present, specifies the only record to process. A value of 0 means to process no records. Overrides nNext and lRest. <lRest> If lExport is .T., then if set to .T. and there are no nRecord, nNext, or bWhile arguments, processes all records from current to last.

Returns

Description

__dbDelim() copies all or selected contents of a database table to an SDF text file or appends all or selected contents of an SDF text file to a database table.

Examples

```
// Copy delinquent accounts into a delimited text file.
USE ACCOUNTS NEW
COPY TO overdue DELIMITED FOR !EMPTY( accounts->duedate ) ;
.AND. DATE() - accounts->duedate > 30
// Import new customer records.
USE CUSTOMER NEW
APPEND FROM customer DELIMITED
```

Tests

Status

Started
__dbDelim() is intended to be fully compliant with CA-Clipper's function of the same name and is the underlying implementation of the APPEND FROM DELIMITED and COPY TO DELIMITED commands.

Platforms

All

Files

See Also:

[__dbSDF()]
[ARRAY()]
[ARRAY()]

__dbSDF()

Copies the contents of a database to an SDF text file or

Syntax

```
__dbSDF( <lExport>, <xcFile>, [<aFields>],
[<bFor>], [<bWhile>], [<nNext>], [<nRecord>], <lRest>  ) --> NIL
```

Arguments

<lExport> If set to .T., copies records to an SDF file. If set to .F., append records from an SDF file. <xcFile> The name of the text file to copy to or append from. If a file extension is not specified, ".txt" is used by default. <aFields> An aray of field names to limit the processint to. If not specified, or if empty, then all fields are processed. <bFor> An optional code block containing a FOR expression that will reduce the number of records to be processed. <bWhile> An optional code block containing a WHILE expression that will reduce the number of records to be processed. <nNext> If present, but nRecord is not present, specifies to process this number of records, starting with the current record. A value of 0 means to process no records. <nRecord> If present, specifies the only record to process. A value of 0 means to process no records. Overrides nNext and lRest. <lRest> If lExport is .T., then if set to .T. and there are no nRecord, nNext, or bWhile arguments, processes all records from current to last.

Returns

Description

__dbSDF() copies all or selected contents of a database table to an SDF text file or appends all or selected contents of an SDF text file to a database table.

Examples

```
// Copy delinquent accounts into an SDF text file.
USE ACCOUNTS NEW
COPY TO overdue SDF FOR !EMPTY( accounts->duedate ) ;
.AND. DATE() - accounts->duedate > 30
// Import new customer records.
USE CUSTOMER NEW
APPEND FROM customer SDF
```

Tests

Status

Started
__dbSDF() is intended to be fully compliant with CA-Clipper's function of the same name and is the underlying implementation of the APPEND FROM SDF and COPY TO SDF commands.

Platforms

All

Files

See Also:

__dbDelim()
ARRAY()
ARRAY()

ADS Overview

Advantage Database Server RDD

Description

RDDADS is an RDD for the Advantage Database Server, an xBase data server by Extended Systems <www.advantagedatabase.com>. The RDD was written by Alexander Kresin <alex@belacy.belgorod.su> Additional code and documentation was added by Brian Hays <bhays@abacuslaw.com>.

Your Harbour application can access a remote database server for a true client/server architecture, or it can use the "local server" ADSLOC32.DLL for stand-alone or even small network installations.

For using this RDD you need to have:
ACE32.DLL (Advantage Client Engine),
AXCWS32.DLL (communication layer for remote server) or
ADSLOC32.DLL (local server)

You need also to create ace32.lib with the help of implib.exe: implib ace32.lib ace32.dll

Then build rddads.lib using make_b32.bat or make_vc.bat.

For building executables don't forget to include the ace32.lib and rddads.lib in the make file or link script.

You also need to include in your PRG file following lines:

```
REQUEST _ADS
rddRegister( "ADS", 1 )
rddsetdefault( "ADS" )
```

By default RDDADS is tuned for remote server and cdx indexes. To change this you may use these commands defined in ads.ch:

```
SET SERVER LOCAL  SET SERVER REMOTE

SET FILETYPE TO NTX  SET FILETYPE TO ADT  SET FILETYPE TO CDX
```

or functions AdsSetServerType(), AdsSetFileType(). See the header file ADS.CH for details.

Note that the default local server (ADSLOC32.DLL) is useable for file sharing on a small network. The default DLL is limited to 5 users, but an unlimited version is available from Extended Systems.

MAX OPEN TABLES: The server (even local) has its own setting for Max Tables allowed open. For the Local Server, it can be set in ADSLOCAL.CFG. The default is only 50! For the Windows Remote Servers, use the Configuration Utility, or increase the setting for the TABLES configuration value in the Advantage Database Server configuration registry key using the Registry Editor. For NetWare, edit the configuration file ADS.CFG.

See ACE.HLP under ADSLOCAL.CFG, or the Advantage Error Guide for error 7005.

Every attempt has been made to make the rdd compliant with the standard dbfcdx rdd at the .PRG level. One important difference is the handling of **structural indexes. ACE will always automatically open an index with the** same name as the data file. There is no way to turn this feature off.

Be sure to use the command SET DEFAULT TO (cDir) and not its equivalent Set() function call. The Set() function will not make the call to ADS to change its internal setting, but the command will. The same is true for DATEFORMAT, DELETE, and EPOCH.

For programmers who are already familiar with the ACE engine, this also means there are some differences between the RDDADS in Harbour and the parallel ACE documentation.

1) In ACE, skipping backwards to BOF goes to the phantom record and sets the record number to 0. In RDDADS, the record pointer stays at the Top record and only the BOF flag is set to True.

2) In RDDADS, a filter expression can be used that may not be valid on the

server (because of references to public variables or User-Defined Functions). In these cases, all data will come back from the server but will be filtered by the application running on the client. These situations lose the benefits of having a data server and should be avoided, but they will function as they would in a Clipper program.

One problem with this scenario is that index key counting functions that are supposed to give an accurate count respecting the filter (e.g. dbOrderInfo(DBOI_KEYCOUNT) will return the values the Server knows about, so the counts will be inaccurate.

3) When setting a relation, the expression must be one that can be evaluated by the Advantage Expression Engine. UDFs will fail.

GT_ASCPOS()

Return the ascii value of a specified character in a string

Syntax

GT_Ascpos(<cStr>, <nPos>) --> nAscVal

Arguments

<cStr>   - The string <nPos> - The position in <cStr>

Returns

<nAscVal> - The ascii value of substr(<cStr>, <nPos>, 1)

Description

Return the ascii value of a specified character in a string Equivalent (but much faster) to asc(substr(cStr, nPos, 1)

NOTE: invalid parameters will return -1 nPos > len(cStr)   will return -2

This last behaviour is different to the Funcky function of the same name. I changed the behaviour because some of the strings I process contain embedded NULs.

Examples

```
? gt_ascpos("the cat sat on the mat", 3) // prints e
```

Status

Ready

Files

Library is libgt

GT_ASCIISUM()

Sum the ascii values in a string.

Syntax

GT_AsciiSum(<cStr>) --> nSum

Arguments

<cStr> - The string to sum

Returns

<nSum> - The sum of all ascii values in <cStr>.

Description

Sum the ascii value of every character in the passed string and return the result.

Status

Ready

Files

Library is libgt

GT_ATDIFF()

Return the position where two strings begin to differ

Syntax

GT_AtDiff(<cStr1>, <cStr2>) --> nPos

Arguments

<cStr1> - A character string to compare <cStr2> - The string to compare with

Returns

<nPos> - The position in <cStr2> where <cStr1> begins to differ

Description

Return the position in <cStr2> where <cStr1> begins to differ. If the strings differ in the first character GT_AtDiff() will return 1. If the two strings are identical (or identical upto the last character in <cStr2>) the function will return 0.

NOTE: invalid parameters will return -1

Examples

```
? gt_atDiff("the cat", "the rat")          // prints 5
? gt_atDiff("the cat", "the ")             // prints 0
```

Status

Ready

Files

Library is libgt

GT_CHAREVEN()
Return a string of all the characters in even positions

Syntax

GT_CharEven(<cStr>) --> cRet

Arguments

<cStr> - A character string to extract chars from

Returns

<cRet> - A string of all the chars in even positions

Description

Return a string consisting of all the characters in even positions in <cStr1>.

NOTE: invalid parameters will return ""

Examples

```
? gt_CharEven("abcdefghijklm")             // prints "bdfhjl"
```

Status

Ready

Files

Library is libgt

GT_CHARMIX()

Amalgamate two strings to form the return value

Syntax

GT_CharMix(<cStr1>, <cStr2>) --> cRet

Arguments

<cStr1> - A character string to mix <cStr2> - A character string to mix with

Returns

<cRet> - A string consisting of all the characters in <cStr1> mixed with all the characters in <cStr2>

Description

Return a string consisting of all the characters in <cStr1> mixed with the characters from <cStr2>.

NOTE: invalid parameters will return ""

Examples

```
? gt_CharMix("abc", "123")              // prints "a1b2c3"
? gt_CharMix("abcde", "123")            // prints "a1b2c3de"
? gt_CharMix("abc", "12345")            // prints "a1b2c345"
```

Status

Ready

Files

Library is libgt

GT_CHARODD()

Return a string of all the characters in odd positions

Syntax

GT_CharOdd(<cStr>) --> cRet

Arguments

<cStr> - A character string to extract chars from

Returns

<cRet> - A string of all the chars in odd positions

Description

Return a string consisting of all the characters in odd positions in <cStr1>.

NOTE: invalid parameters will return ""

Examples

```
? gt_CharOdd("abcdefghijklm")             // prints "acegikm"
```

Status

Ready

Files

Library is libgt

GT_CHRCOUNT()

Count the number of times a character appears in a string

Syntax

GT_ChrCount(<cChr>, <cStr>) --> nFreq

Arguments

<cChr> - The character to find the frequence of <cStr> - The string in which to find the character

Returns

Description

GT_ChrCount() counts how many times a specified character appears in a string.

NOTE: invalid parameters will return -1

Examples

```
? GT_ChrCount("t", "the cat sat on the mat")      // prints 4
```

Status

Ready

Files

Library is libgt

GT_CHRFIRST()

Find which character occurs first in a string

Syntax

GT_ChrFirst(<cChars>, <cStr>) --> nAsc

Arguments

<cChars> - The set of characters to find <cStr> - The input string

Returns

<nAsc> - The ASCII value of the first character in <cChars> which appears first in <cStr>

Description

Return the ascii value of a character in <cChars> which appears first in <cStr>.

Examples

```
? chr(GT_ChrFirst("sa ", "This is a test"))  // prints "s"
? chr(GT_ChrFirst("et",  "This is a test"))   // prints "t"
```

Status

Ready

Files

Library is libgt

GT_CHRTOTAL()

Find number of times a set of characters appears in a string

Syntax

GT_ChrTotal(<cChrs>, <cStr>) --> nTotOcc

Arguments

<cChrs> - The set of characters <cStr> - The string to search

Returns

<nTotOcc> - The number of times the characters specified in <cChrs> appears in <cStr>

Description

Returns the numnber of occurrences of characters belonging to the set <cChrs> in the string <cStr>. If no characters in <cChrs> appears in <cStr> GT_ChrTotal() will return 0.

NOTE: invalid parameters will return -1

Examples

```
local cStr1 := "the cat sat on the mat"

? GT_ChrTotal("tae", cStr1)            // prints 10
? GT_ChrTotal("zqw", cStr1)            // prints  0
```

Status

Ready

Files

Library is libgt

GT_STRCOUNT()

Count the number of times a substring appears in a string

Syntax

GT_StrCount(<cChrs>, <cStr>) --> nFreq

Arguments

<cChrs> - The substring to find the frequence of <cStr> - The string in which to find the character

Returns

<nFreq> - The number of times <cChrs> occurs in <cStr>

Description

GT_StrCount() counts how many times a specified substring appears in a string. If the substring does NOT appear in <cStr> this function will return 0. If the substring is a single character use GT_ChrCount() as it will be faster.

NOTE: invalid parameters will return -1

Examples

```
? GT_StrCount("the", "the cat sat on the mat")      // prints 2
```

Status

Ready

Files

Library is libgt

GT_STRCSPN()

Return length of prefix in string of chars NOT in set.

Syntax

GT_strcspn(<cString>, <cSet>) --> nLength

Arguments

<cString> - The string to find the prefix in <cSet> - The set of characters

Returns

<nLength> - The length of a string upto a character in the set

Description

Return the number of characters in the leading segment of a string that consists solely of characters NOT in the set.

Examples

```
? GT_strcspn("this is a test", "as ")      // prints 3
? GT_strcspn("this is a test", "elnjpq")   // prints 11
```

Status

Ready

Files

Library is libgt

GT_STRDIFF()

Return a string where it begins to differ from another

Syntax

```
GT_StrDiff(<cStr1>, <cStr2>) --> cRet
```

Arguments

<cStr1>   - A character string to compare <cStr2> - The string to compare with

Returns

<cRet>   - A string beginning at the position in <cStr2> where <cStr1> begins to differ from <cStr1>

Description

Return a string beginning at the position in <cStr2> where <cStr1> begins to differ from <cStr1>. If the two strings are identical (or identical upto the last character in <cStr2>) the function will return "".

NOTE: invalid parameters will return ""

Examples

```
? gt_strDiff("the cat", "the rat")          // prints "rat"
? gt_strDiff("the cat", "the ")             // prints ""
```

Status

Ready

Files

Library is libgt

GT_STREXPAND()

Insert fillers between characters in a passed string

Syntax

GT_StrExpand(<cStr>, [<nNum>], [<cChar>]) --> cRet

Arguments

<cStr1> - A character string to insert chars into <nNum> - The number of fill characters to insert (default 1) <cChar> - The fill chararacter (default space)

Returns

<cRet> - The input string with fill characters inserted between every character in the original.

Description

Inserts fill characters into a string.

NOTE: invalid parameters will return ""

Examples

```
? gt_strexpand("abc")                    // prints "a b c"
? gt_strexpand("abc", 2)                 // prints "a  b  c"
? gt_strexpand("abc", 2, 'þ')            // prints "aþþbþþc"
```

Status

Ready

Files

Library is libgt

GT_STRLEFT()

Find length of prefix of a string

Syntax

GT_StrLeft(<cStr>, <cChars>) --> nLen

Arguments

<cStr> - The input string <cChars> - The set of characters to find

Returns

Description

Return the length of the leading segment in the passed string <cStr> that consists solely of the characters in the character set <cChars>.

If no characters in the the search set are found, the function shall return 0

Examples

```
? GT_StrLeft("this is a test", "hsit ")       // prints 8
? GT_StrLeft("this is a test", "hit a")       // prints 3
? GT_StrLeft("this is a test", "zxy")         // prints 0
```

Status

Ready

Files

Library is libgt

GT_STRPBRK()

Return string after 1st char from a set

Syntax

GT_StrpBrk(<cStr>, <cSet>) --> cString

Arguments

<cStr> - The input string <cSet> - The set of characters to find

Returns

<cString> - The input string after the first occurance of any character from <cSet>

Description

Return a string after the first occurance of any character from the input set <cSet>.

Examples

```
? GT_Strpbrk("This is a test", "sa ")  // prints "s is a test"
? GT_Strpbrk("This is a test", "et")   // prints "test"
```

Status

Ready

Files

Library is libgt

GT_STRRIGHT()

Find length of a suffix of a string

Syntax

GT_StrRight(<cStr>, <cChars>) --> nLen

Arguments

<cStr> - The input string <cChars> - The set of characters to find

Returns

<nLen> - The length of the prefix found.

Description

Return the length of the trailing segment in the passed string <cStr> that consists solely of the characters in the character set <cChars>.

If no characters in the the search set are found, the function shall return 0

Examples

```
? GT_StrRight("this is a test", "teas ")       // prints 8
? GT_StrRight("this is a test", "tes h")       // prints 5
? GT_StrRight("this is a test", "zxy")         // prints 0
```

Status

Ready

Files

Library is libgt

GT_NEWFLAG()

Create a new bit flag string.

Syntax

GT_NewFlag(<nFlagCount>) --> cFlagString

Arguments

<nFlagCount> is the number of flags you wish to store.

Returns

Description

GT_NewFlag() is used to construct a bit flag string. The bit flag functions can be used for storing a large number of logical values in a small space.

To create a bit flag string you need to pass GT_NewFlag() a value that is equal to or greater than the number of flags required (you may want to allow for future expansion). Each character in the string returned from GT_NewFlag() will hold 8 logical values.

Examples

```
cFlags := GT_NewFlag(20)   // Create a bit flag string for 20
                           // logical values.
```

See Also:

ARRAY()

GT_SETFLAG()

Set a number of flags to TRUE in a bit flag string.

Syntax

GT_SetFlag(<cFlagString>,[<nStart>],[<nEnd>]) --> cFlagString

Arguments

<cFlagString> is a bit flag string created with GT_NewFlag()

<nStart> is the starting flag. This is an optional numeric value. If not supplied it defaults to 1.

<nEnd> is the ending flag. This is an optional numeric value. If not supplied it defaults to <nStart>.

Returns

Description

GT_SetFlag() is used to turn flags within the flag string on.

Examples

```
cFlags := GT_NewFlag(20)   // Create a bit flag string for 20
                           // logical values.

// Now set flags 10 to 15 to true.

cFlags := GT_SetFlag(cFlags,10,15)

// And set flag 18 to true.

cFlags := GT_SetFlag(cFlags,18)

// And set flag 1 to true.

cFlags := GT_SetFlag(cFlags)
```

See Also:

ARRAY()

GT_CLRFLAG()

Set a number of flags to FALSE in a bit flag string.

Syntax

GT_ClrFlag(<cFlagString>,[<nStart>],[<nEnd>]) --> cFlagString

Arguments

<cFlagString> is a bit flag string created with GT_NewFlag()

<nStart> is the starting flag. This is an optional numeric value. If not supplied it defaults to 1.

<nEnd> is the ending flag. This is an optional numeric value. If not supplied it defaults to <nStart>.

Returns

Description

GT_ClrFlag() is used to turn flags within the flag string off.

Examples

```
cFlags := GT_NewFlag(20)   // Create a bit flag string for 20
                           // logical values.

// Now, turn them all on.

cFlags := GT_SetFlag(cFlags,1,20)

// Now set flags 10 to 15 to false.

cFlags := GT_ClrFlag(cFlags,10,15)

// And set flag 18 to false.

cFlags := GT_ClrFlag(cFlags,18)

// And set flag 1 to false.

cFlags := GT_ClrFlag(cFlags)
```

See Also:

ARRAY()

GT_ISFLAG()

Test the setting of a flag in a bit flag string.

Syntax

GT_IsFlag(<cFlagString>,[<nFlag>]) --> lSetting

Arguments

<cFlagString> is a bit flag string created with GT_NewFlag()

<nFlag> is the flag to be tested.

Returns

Description

GT_IsFlag() is used to test the state of a flag with a bit flag string.

Examples

```
// Print the setting of the flags in a flag string called ``cDave''

for nFlag := 1 to (len(cDave)*8)
   ? "Flag number ",nFlag," == ",GT_IsFlag(cDave,nFlag)
next
```

See Also:

ARRAY()

TFileRead()

Read a file one line at a time

Syntax

oFile := TFileRead():New(<cFileName> [, <nReadSize>])

Arguments

<cFileName> is the required name of the file to be read.

<nReadSize> is the optional size to use when reading from the file. The default value is 4096 and the allowed range is 1 through 65535. Any value outside of this range causes the default value to be used.

Returns

Description

TFileRead() is used to access a file one line at a time. You must specify the name of the file when an instance of the class is created.
The class data should be considered private to the class.

The class methods are as follows:

New()   Creates a new instance of the TFileRead class.

Open([<nFlags>])   Opens the file for reading. The optional nFlags parameter can use any of the FOPEN() flags from fileio.ch. The default is FO_READ + FO_SHARED. Calling this method when the file is already open causes the next ReadLine() to start over from the beginning of the file.

Close()   Closes the file.

ReadLine()   Returns one line from the file, stripping the newline characters. The following sequences are treated as one newline: 1) CR CR LF; 2) CR LF; 3) LF; and 4) CR. Note: LF CR is 2 newlines.
Name()   Returns the name of the file.

IsOpen()   Returns .T. if the file is open.

MoreToRead()   Returns .T. if there are more lines to be read (think of it as an inverse EOF function).

Error()   Returns .T. if an error has occurred.

ErrorNo()   Returns the current error code.

ErrorMsg([<cPre>]) Returns a formatted error message.

Examples

```
#ifdef __HARBOUR__
 #define NEW_LINE CHR( 10 )
#else
 #define NEW_LINE CHR( 13 ) + CHR( 10 )
#endif
#include "fileio.ch"

PROCEDURE Main( cFile )
LOCAL oFile := TFileRead():New( cFile )

   oFile:Open()
   IF oFile:Error()
      QOUT( oFile:ErrorMsg( "FileRead: " ) )
   ELSE
      WHILE oFile:MoreToRead()
         OUTSTD( oFile:ReadLine() )
         OUTSTD( NEW_LINE )
      END WHILE
      oFile:Close()
   END IF
QUIT
```

Tests

See Examples

Status

Ready

Compliance

This is a new Harbour Tools class

Files

Library is libmisc

See Also:

ARRAY()

ISBIN()

Check if the value is a Binary Number

Syntax

ISBIN(<cN>) -><cNr>

Arguments

<cN> STRING TO BE CHECKED

Returns

<cNr> .T. IF THE STRING IS BYNARY,otherwise .F.

Description

check if the passed string is a bynary number or not

Files

Library is libmisc

See Also:

ISOCTAL()
ISDEC()
ISHEXA()

ISOCTAL()

Check if the value is a Octal Number

Syntax

ISOCTAL(<cN>) -><cNr>

Arguments

<cN> STRING TO BE CHECKED

Returns

<cNr> .T. IF THE STRING IS OCTAL;otherwise .F.

Description

check if the passed string is a octal number or not

Files

Library is libmisc

See Also:

ISBIN()
ISDEC()
ISHEXA()

ISDEC()

Check if the value is a Decimal Number

Syntax

ISDEC(<cN>) -><cNr>

Arguments

<cN> STRING TO BE CHECKED

Returns

<cNr> .T. IF THE STRING IS DECIMAL;otherwise .F.

Description

check if the passed string is a decimal number or not

Files

Library is libmisc

See Also:

ISOCTAL()
ISBIN()
ISHEXA()

ISHEXA()

Check if the value is a Hexal Number

Syntax

ISHEXA(<cN>) -><cNr>

Arguments

<cN> STRING TO BE CHECKED

Returns

<cNr> .T. IF THE STRING IS HEXA;otherwise .F.

Description

check if the passed string is a hexa number or not

Files

Library is libmisc

See Also:

ISOCTAL()
ISDEC()
ISBIN()

DECTOBIN()

Converts a Decimal Value to Binary

Syntax

DECTOBIN(<cN>) -><cNr>

Arguments

<cN> NUMBER TO BE CONVERTED

Returns

<cNr> NUMBER CONVERTED

Description

This function converts a string <cN> from an decimal value to an binary value.

Files

Library is libmisc

See Also:

DECTOHEXA()
DECTOOCTAL()

DECTOOCTAL()

Converts a Decimal Value to Octal

Syntax

DECTOOCTAL(<cN>) -><cNr>

Arguments

<cN>	NUMBER TO BE CONVERTED

Returns

<cNr>	NUMBER CONVERTED

Description

This function converts a string <cN> from an decimal value to an octal value.

Files

Library is libmisc

See Also:

DECTOHEXA()
DECTOBIN()

DECTOHEXA()

Converts a Decimal Value to Hexa

Syntax

DECTOHEXA(<cN>) -><cNr>

Arguments

<cN>	NUMBER TO BE CONVERTED

Returns

<cNr>	NUMBER CONVERTED

Description

This function converts a string <cN> from an decimal value to an hexadecimal value.

Files

Library is libmisc

See Also:

DECTOBIN()
DECTOOCTAL()

BINTODEC()

Converts a Binary Value to Decimal

Syntax

```
BIntODEC(<cN>) -><cNr>
```

Arguments

<cN>	NUMBER TO BE CONVERTED

Returns

<cNr>	NUMBER CONVERTED

Description

This function converts a string <cN> from an binary value to a numeric decimal value.

Files

Library is libmisc

See Also:

OCTALTODEC()
HEXATODEC()

OCTALTODEC()

Converts a Octal Value to Decimal

Syntax

```
OCTALTODEC(<cN>) -><cNr>
```

Arguments

<cN> NUMBER TO BE CONVERTED

Returns

<cNr> NUMBER CONVERTED

Description

This function converts a string <cN> from an octal value to a numeric decimal value.

Files

Library is libmisc

See Also:

BINTODEC()
HEXATODEC()

HEXATODEC()

Converts a Hexa Value to Decimal

Syntax

HEXATODEC(<cN>) -><cNr>

Arguments

<cN> NUMBER TO BE CONVERTED

Returns

<cNr> NUMBER CONVERTED

Description

This function converts a string <cN> from an hexadecimal value to a numeric decimal value.

Files

Library is libmisc

See Also:

OCTALTODEC()
BINTODEC()

FIELDTYPE()

Determines the type of a given field.

Syntax

FIELDTYPE(<nFieldNum>) --> cFieldType

Arguments

<nFieldNum> Data field , which type need to be determined.

Returns

FIELDTYPE() returns the character that designates the type of a given field:

'C'	character string;
'N'	numeric;
'L'	logical;
'D'	date;
'M'	memo.

Description

This function determines the type of a field, designated by its number.

Examples

```
FUNCTION Main()
LOCAL i
USE Tests NEW
FOR i = 1 TO FCOUNT()
  ? FieldType( i )
NEXT
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-CLIPPER tools compatible

Files

Library is libmisc

See Also:

FIELDSIZE()
FIELDDECI()

FIELDSIZE()

Determines the size of a given field.

Syntax

```
FIELDSIZE(<nFieldNum>) --> nFieldSize
```

Arguments

<nFieldNum> Data field , which size need to be determined.

Returns

FIELDSIZE() returns the number that designates the size of a given field.

Description

This function determines the size of a field, designated by its number.

Examples

```
FUNCTION Main()
LOCAL i
USE Tests NEW
FOR i = 1 TO FCOUNT()
  ? FieldSize( i )
NEXT
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-CLIPPER tools compatible

Files

Library is libmisc

See Also:

FIELDTYPE()
FIELDDECI()

FIELDDECI()

Determines the number of decimal places of a given numeric field.

Syntax

FIELDDECI(<nFieldNum>) --> nFieldDeci

Arguments

<nFieldNum> Numeric data field , for which number of decimal places need to be determined.

Returns

FIELDDECI() returns the numeric value that designates the number of decimal places of a given field.

Description

This function determines the number of decimal places of a given numeric field.

Examples

```
FUNCTION Main()
LOCAL i
USE Tests NEW
FOR i = 1 TO FCOUNT()
  ? FieldDeci( i )
NEXT
USE
RETURN NIL
```

Status

Ready

Compliance

This function is CA-CLIPPER tools compatible

Files

Library is libmisc

See Also:

FIELDTYPE()
FIELDSIZE()

THtml()

Html Class

Syntax

oHtml:=THtml():New(<cFile>) --> oHtm

Arguments

<cFile> Name of the Html file to create

Returns

<oHtm> An instance of the THtml Class

Description

THtml() is a class that creates an .html file of the same name you pass to the constructor.
The class methods are as follows:

```
New(<cFile>)          Create a new instance of the THtml class
Close()               Close the created file
WriteTitle(<cTitle>) Write the file title
WritePar(<cPar>)      Writes a paragraph
WriteParBold(<cPar>) Same as WritePar(), but the text is bold
WriteLink(<cLink>,<cName>)  Write a link to another topic
WriteText(<cText>)    Write any text
```

Examples

```
FUNCTION MAIN()

LOCAL oHtm

oHtm := THTML():New( "www\harbour.html" )
oHtm:WriteTitle( "Harbour Reference Guide" )
oHtm:WritePar( "HARBOUR" )
oHtm:WriteLink( "OverView" )
oHtm:WriteLink( "License" )
oHtm:WriteLink( "http://www.gnu.org/copyleft/gpl" )
oHtm:WritePar( "See the Links Above" )
oHtm:Close()
RETURN Nil
```

</par>

Status

Ready

Compliance

This is a new Harbour Tools class

Platforms

ALL

See Also:

ARRAY()

TOs2()

OS/2 Documentation Class

Syntax

oNg:=TOs2():New(<cFile>) --> oOs2

Arguments

<cFile> Name of the IPF Source file to create

Returns

<oOs2> An instance of the TOs2 Class

Description

TOs2() is a class that creates the OS/2 IPF Source of the same name you pass to the constructor.
The class methods are as follows:

```
New(<cFile>)           Create a new instance of the TOs2 class
Close()                Close the created file
WriteTitle(<cTopic>,<cTitle>)  Write the file title
WritePar(<cPar>)       Write a paragraph
WriteParBold(<cPar>)   Same as WritePar(), but the text is bold
WriteLink(<cLink>)     Write a link to another topic
ScanLink(<clink>)      Scan the aLinkRef array for a valid topic
DosToOs2Text(<cText>) Convert a Dos string to a OS/2 String
```

Examples

```
FUNCTION MAIN()

LOCAL oNg

oNg := TOs2():New( "ngi\harbour.ngi" )
oNg:WriteTitle( "Harbour Reference Guide" )
oNg:WritePar( "HARBOUR" )
oNg:WriteLink( "OverView" )
oNg:WriteLink( "License" )

oNg:WritePar( "See the Links Above" )
oNg:Close()
RETURN Nil
```

Status

Ready

Compliance

This is a new Harbour Tools class

Platforms

ALL

See Also:

TNortonGuide()

TNortonGuide()
Norton Guide Class

Syntax

oNg:=TNortonGuide():New(<cFile>) --> oNg

Arguments

<cFile> Name of the Ng Source file to create

Returns

<oNg> An instance of the TNortonGuide Class

Description

TNortonGuide() is a class that creates the Norton Guide Source Code of the same name you pass to the constructor.
The class methods are as follows:

```
New(<cFile>)           Create an instance of the TNortonGuide class
Close()                Close the created file
WriteTitle(<cTopic>,<cTitle>)  Write the file title
WritePar(<cPar>)       Write a paragraph
WriteParBold(<cPar>) Same as WritePar(), but the text is bold
WriteLink(<cLink>)     Write a link to another topic
```

Examples

```
FUNCTION MAIN()

LOCAL oNg

oNg := TNortonGuide():New( "ngi\harbour.ngi" )
oNg:WriteTitle( "Harbour Reference Guide" )
oNg:WritePar( "HARBOUR" )
oNg:WriteLink( "OverView" )
oNg:WriteLink( "License" )

oNg:WritePar( "See the Links Above" )
oNg:Close()
RETURN Nil
```

Status

Ready

Compliance

This is a new Harbour Tools class

Platforms

ALL

See Also:

TTroff()
TRtf()
THtml()
TOs2()

TRtf()

Rtf Class

Syntax

oNg:=TRtf():New(<cFile>) --> oRtf

Arguments

<cFile> Name of the RTF file to create

Returns

<oRtf> An instance of the TRtf Class

Description

TRtf() is a class that creates the RTF Documentation Source Code of the same name you pass to the constructor.
The class methods are as follows:

```
New(<cFile>)          Create a new instance of the TRtf class
Close()               Close the create file
WriteTitle(<cTopic>,<cTitle>)  Write the file title
WritePar(<cPar>)      Write a paragraph
WriteParBold(<cPar>) Same as WritePar(), but the text is bold
WriteLink(<cLink>)    Write a link to another topic
WriteHeader()         Write the RTF header
EndPar()              Write the end paragraph delimiter
```

Examples

```
FUNCTION MAIN()

LOCAL oRtf

oRtf := TRtf():New( "rtf\harbour.rtf" )
oRtf:WriteHeader()
oRtf:WriteTitle( "Harbour Reference Guide" )
oRtf:WritePar( "HARBOUR" ):Endpar()
oRtf:WriteLink( "OverView" )
oRtf:WriteLink( "License" )

oRtf:WritePar( "See the Links Above" ):EndPar()
oRtf:Close()
RETURN Nil
```

Status

Ready

Compliance

This is a new Harbour Tools class

Platforms

ALL

See Also:

TNortonGuide()

TTroff()

Troff Class

Syntax

oTroff:=TTrof():New(<cFile>) --> oTrf

Arguments

<cFile> Name of the Troff file to create

Returns

<oTrf> instance of the TTroff Class

Description

TTroff() is a class that creates the TROFF Documentation Source Code of the same name you pass to the constructor.
The class methods are as follows:

```
New(<cFile>)          Create a new instance of the THtml class  Close()
Close the created file
WriteTitle(<cTopic>,<cTitle>) Write the file title
WritePar(<cPar>)      Write a paragraph
WriteParBold(<cPar>) Same as WritePar(), but the text is bold
WriteLink(<cLink>)    Write a link to another topic
WriteText()           Writes text without formating
```

Examples

```
FUNCTION MAIN()

LOCAL oTroff
oTroff := TTroff():New( "tr\harbour.ngi" )
oTroff:WriteTitle( "Harbour Reference Guide" )
oTroff:WritePar( "HARBOUR" )
oTroff:WriteLink( "OverView" )
oTroff:WriteLink( "License" )

oTroff:WritePar( "See the Links Above" )
oTroff:Close()

RETURN Nil
```

Status

Ready

Compliance

This is a new Harbour Tools class

Platforms

ALL

See Also:

TNortonGuide()

CD()

Change the Current Directory

Syntax

CD(<cDir>) --> lSuccess

Arguments

<cDir> DIR TO BE CHANGED

Returns

<lSucess> .T. IF SUCESSFUL; otherwise .F.

Description

CHANGE THE CURRENT DIRECTORY

Examples

```
IF CD("OLA")
   RETURN(.T.)
ELSE
   RETURN(.F.)
ENDIF
```

Files

Header is Fileio.ch

See Also:

MD()
RD()

MD()

Creates a Directory

Syntax

MD(<cDir>) -> <lSucess>

Arguments

<cDir> DIRECTORY TO BE CREATED

Returns

<lSucess> .T. IF SUCESSFUL; otherwise .F.

Description

CREATE A DIRECTORY

Examples

```
IF MD("OLA")
  RETURN(.T.)
ELSE
  RETURN(.F.)
ENDIF
```

Files

Header is Fileio.ch

See Also:

CD()
MD()

RD()
Remove a Directory

Syntax

RD(<cDir>) --> <lSucess>

Arguments

<cDir> DIR TO BE DELETED

Returns

<lSucess> .T. IF SUCESSFUL; otherwise .F.

Description

REMOVE A DIRECTORY

Examples

```
IF RD("OLA")
  RETURN(.T.)
ELSE
  RETURN(.F.)
ENDIF
```

Files

Header is Fileio.ch

See Also:

CD()
MD()

StrFormat()

Format a string

Syntax

```
StrFormat(<cMask>[, <cPar1>[, <cParn>[, ...]]]) --> cString
```

Arguments

<cMask> Holds the mask for the resulting string
<cParn> Holds the strings to be inserted in the mask maximum 9 of them can be specified.

Returns

<cString> Return the mask with all the parameters inserted.

Description

String replacment, can be useful when writing international apps. You can separate the constant strings from the variable ones. Each %1 - %9 marks will be replaced with the appropriate parameter from the parameter list.
Marks can be in any order, and can be duplicated.
You can print "%" character with "%%".

Examples

```
StrFormat("Please insert disk %1 to drive %2", LTrim(Str(2)), "A:")
StrFormat("This is %1 from %2", "Victor", "Hungary")
StrFormat("%2 %1 %2", "Param1", "Param2")
```

Tests

```
? StrFormat("Please insert disk %1 to drive %2", LTrim(Str(2)), "A:")
? StrFormat("This is %1 from %2", "Victor", "Hungary")
? StrFormat("%2 %1 %2", "Param1", "Param2")
? StrFormat("Hello")
? StrFormat("%1 - %2", "one")
? StrFormat("%1 - %2", "one", "two")
? StrFormat("%2 - %1", "one", "two")
? StrFormat("%2 - %", "one", "two")
? StrFormat("%% - %", "one", "two")
? StrFormat("%9 - %", "one", "two")
```

Status

Done

Compliance

All platforms

Files

Library is libmisc

AMONTHS()

Returns an array with the months names.

Syntax

```
AMONTHS()  --> aMonths
```

Arguments

Returns

<aMonths> The array which holds the months names.

Description

This function returns an array with all the months names in the selected current language.

Examples

```
aMonths := AMonths()
? aMonths[1] -> January
? aMonths[1] -> Enero (if the selected language is Spanish)
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

ADAYS()

ADAYS()

Returns an array with the days names.

Syntax

```
ADAYS()  --> aDays
```

Arguments

Returns

<aDays>   The array which holds the days names.

Description

This function returns an array with all the days names in the selected current language.

Examples

```
aDays := ADays()
? aDays[1] -> Sunday
? aDays[1] -> Domingo (if the selected language is Spanish)
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

ADAYS()

ISLEAPYEAR()

Checks if the given date is a leap year.

Syntax

```
ISLEAPYEAR( <dDate> )  --> lTrueOrFalse
```

Arguments

<dDate> A valid date.

Returns

<lTrueOrFalse> A logical that indicates if the date year is leap

Description

This function returns true if the given date is a leap year and false if isn't.

Examples

```
? IsLeapYear( DToC( "01/01/2000" ) ) -> .t.
? IsLeapYear( DToC( "01/01/2001" ) ) -> .f.
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

DAYSINMONTH()

DAYSINMONTH()

Gets the days in a month.

Syntax

```
DAYSINMONTH( <dDate> )  --> nDays
```

Arguments

<dDate> A valid date.

Returns

<nDays> The number of days of the month.

Description

This function returns the number of days of the given date month.

Examples

```
? DaysInMonth( DToC( "01/01/2000" ) ) -> 31
? DaysInMonth( DToC( "02/01/2000" ) ) -> 29
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

ISLEAPYEAR()

EOM()

Gets the last day in a month.

Syntax

```
EOM( <dDate> )  --> dEOM
```

Arguments

<dDate>	A valid date.

Returns

<dEOM>	The last day in the month.

Description

This function returns the last day of a given month date.

Examples

```
? EOM( DToC( "01/01/2000" ) ) -> "01/31/2000"
? EOM( DToC( "02/01/2000" ) ) -> "01/29/2000"
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

BOM()
ARRAY()

BOM()

Gets the first day in a month.

Syntax

```
BOM( <dDate> )  --> dBOM
```

Arguments

<dDate>	A valid date.

Returns

<dBOM>	The first day in the month.

Description

This function returns the first day of a given month date.

Examples

```
? BOM( DToC( "01/25/2000" ) ) -> "01/01/2000"
? BOM( DToC( "02/24/2000" ) ) -> "02/01/2000"
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

EOM()
ARRAY()

DOY()

Gets the day number of the year.

Syntax

```
DOY( <dDate> )  --> nDay
```

Arguments

<dDate>	A valid date.

Returns

<nDay>	The day number

Description

This function returns the day number of the year for a given date.

Examples

```
? DOY( DToC( "01/31/2000" ) ) -> 31
? DOY( DToC( "02/20/2000" ) ) -> 51
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

WOY()

WOY()

Gets the week number of the year.

Syntax

```
WOY( <dDate>, <lIso> )  --> nWeek
```

Arguments

<dDate>   A valid date.

Returns

<nWeek>   The week number <lIso>   Flag that indicates if <nWeek> is in ISO format.

Description

This function returns the week number of the year for a given date. It returns the week number in ISO format (range 0 - 52, by default or passing TRUE as second parameter) or 1 - 52 if lIso is FALSE.

Examples

```
? WOY( DToC( "01/31/2000" ) ) -> 3
? WOY( DToC( "01/31/2000" ), FALSE ) -> 4
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

DOY()

EOY()

Gets the last date of the year.

Syntax

```
EOY( <dDate> )  --> dEOY
```

Arguments

<dDate>	A valid date.

Returns

<dEOY>	The last date of the year.

Description

This function returns the last date of a given year date.

Examples

```
? EOY( DToC( "01/01/2000" ) ) -> "31/12/2000"
? EOY( DToC( "01/01/2001" ) ) -> "31/12/2001"
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

BOY()

BOY()

Gets the first date of the year.

Syntax

```
BOY( <dDate> )  --> dBOY
```

Arguments

<dDate>   A valid date.

Returns

<dBOY>   The first day in the year.

Description

This function returns the first date of a given year date.

Examples

```
? BOY( DToC( "01/25/2000" ) ) -> "01/01/2000"
? BOY( DToC( "02/24/2001" ) ) -> "01/01/2001"
```

Status

Ready

Compliance

This function is new in Harbour.

Platforms

All

Files

Library is libmisc

See Also:

EOY()

ADSBlob2File()

Write a Binary (memo) field's contents to a file

Syntax

```
ADSBlob2File(cFileName, cFieldName) --> lSuccess
```

Arguments

<cFileName> File to create. If it already exists, it will be overwritten on success and destroyed on error.

<cFieldName> Field in the current workarea that contains binary data.

Returns

<lSuccess> True if the file is successfully written.

Description

See ACE.HLP for full details about the Advantage Database Server.
ADSBlob2File() is a wrapper for AdsBinaryToFile.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit only

Files

Library is RddAds Header is ads.ch

See Also:

ADSFile2Blob()

ADSFile2Blob()

Save a Binary file to a field

Syntax

ADSFile2Blob(cFileName, cFieldName, <nBinaryType>) --> lSuccess

Arguments

<cFileName> File to read. Can be in UNC format. A common example is an image file.

<cFieldName> Field in the current workarea to contain the binary data.

<nBinaryType> Either ADS_BINARY (the default) or ADS_IMAGE. This parameter is for fields in DBF files. ADT tables cannot store binary and image data in standard character memo fields (they have specific field types for that).

Returns

<lSuccess> True if the file is successfully written.

Description

See ACE.HLP for full details about the Advantage Database Server. ADSFile2Blob() is a wrapper for AdsFileToBinary. Use of this function is illegal in an ADS transaction.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit only

Files

Library is RddAds Header is ads.ch

See Also:

ADSBlob2File()

ADSClearAOF()

Clears an Advantage Optimized Filter in the current workarea.

Syntax

```
ADSClearAOF()
```

Arguments

Returns

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

- ADSCustomizeAOF()
- ADSEvalAOF()
- ADSGetAOF()
- ADSGetAOFoptLevel()
- ADSIsRecordInAOF()
- ADSRefreshAOF()

ADSCustomizeAOF()

Add or remove records from an existing AOF

Syntax

ADSCustomizeAOF([<nRecno | aRecNos>] [, <nType>]) --> nSuccess

Arguments

<nRecno | aRecNos> Can be either a single record number or an array of record numbers to add or delete from the AOF. If omitted, defaults to the current record.

<nType> The type of operation:

ADS_AOF_ADD_RECORD	Add the record to the AOF (set the bit). This is the default operation.
ADS_AOF_REMOVE_RECORD	Remove the record from the AOF (clear the bit).
ADS_AOF_TOGGLE_RECORD	Switch the record into or out of the AOF.

Returns

<nError> ADS error code, or 0 for success.

Description

An Advantage Optimized Filter (AOF) consists of a bitmap of the records in the database. If bit 5 is on, record 5 is considered a visible record. If bit 5 is off, record 5 is not visible. It does not "pass the test". Initially, the bits are set by the Server according to a filter expression from SET FILTER TO or adsSetAOF(). But by using ADSCustomizeAOF() you can add or remove records at will from the visible set. This is useful for tagging records or for refining a result set after the data has been retrieved from the server.

The maximum number of records that can be customized in a single call is 16,383, so <aRecNos> must not be longer than this.

Calls to AdsCustomizeAOF must be made after an application has created a filter with a call to AdsSetAOF. To create a completely empty record set (to which records can be added with calls to AdsCustomizeAOF), use ".F." as the filter expression given to AdsSetAOF. To create a completely full record set (from which records can be removed), use ".T." as the filter expression.

WARNING: Always start with a FULLY optimized AOF! If an application must use a filter expression that is not fully optimized as the starting point for customization, the ADS_RESOLVE_IMMEDIATE option should be used with the call to AdsSetAOF. Otherwise, the dynamic filter resolution that occurs on the server will automatically remove records that have been added through the AdsCustomizeAOF calls. The filter expressions ".T." and ".F." both result in fully optimized AOFs regardless of available indexes.

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSEvalAOF()

ADSGetAOF()
ADSGetAOFoptLevel()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSEvalAOF()

Evaluate a filter expression to determine its optimization level

Syntax

ADSEvalAOF(<cFilter>) --> nOptimizationLevel

Arguments

<cFilter>	Expression to test.

Returns

ADS_OPTIMIZED_NONE. IMPORTANT NOTE: These values are NOT the same as those returned by dbOrderInfo().

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSGetAOF()
ADSGetAOFoptLevel()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSGetAOFoptLevel()

Returns optimization level of the current AOF filter

Syntax

```
ADSGetAOFoptLevel() --> nOptimizationLevel
```

Arguments

Returns

ADS_OPTIMIZED_NONE.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSGetAOF()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSGetAOF()

Retrieve the filter expression used in the call to AdsSetAOF

Syntax

```
ADSGetAOF() --> cFilter
```

Arguments

Returns

<cFilter> The filter expression used in the call to AdsSetAOF.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSGetAOFoptLevel()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSGetAOFnoOpt()

Return the non-optimized portion of the current filter expression

Syntax

```
ADSGetAOFnoOpt() --> cFilterFragment
```

Arguments

Returns

<cFilterFragment> If an AOF filter expression is not fully optimizable, the non-optimizable part of the expression can be retrieved with this function.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSRefreshAOF()

Update the filter snapshot

Syntax

```
ADSRefreshAOF()
```

Arguments

Returns

Description

See ACE.HLP for full details about the Advantage Database Server. If record updates occur after an AOF is set, the updated records may or may not be valid records for the filter. ADSRefreshAOF() re-evaluates the data to include or exclude changed records.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

- ADSClearAOF()
- ADSIsRecordInAOF()
- ADSSetAOF()

ADSSetAOF()

Create an Advantage Optimized Filter

Syntax

```
ADSSetAOF( <cFilter> [, <nResolveOption>] ) --> lSuccess
```

Arguments

<cFilter> Filter expression to set.

<nResolveOption> Option to indicate how the filter should be resolved in the event that the expression cannot be fully optimized. Options are defined in ads.ch: ADS_RESOLVE_IMMEDIATE, ADS_RESOLVE_DYNAMIC.

Returns

<lSuccess> True if AOF is created.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSIsRecordInAOF()
ADSRefreshAOF()

ADSIsRecordInAOF()

Determine if a record is in the current AOF

Syntax

```
ADSIsRecordInAOF( [<nRecNo>] ) --> lSatisfiesFilter
```

Arguments

<nRecNo>	Record number to test. Default is current record.

Returns

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSClearAOF()
ADSRefreshAOF()

ADSGetRelKeyPos()

Estimated key position of current record within the specified index

Syntax

ADSGetRelKeyPos([<xTag>]) --> nKeyPos

Arguments

<xTag> Index to use. Default is current index.

Returns

position of the current key in the indicated index order. The value returned is between 0.0 and 1.0, inclusive. If there are scopes set on the index order, the position returned is relative to the visible records.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSKeyNo()
ADSKeyCount()

ADSKeyCount()

Retrieve the number of keys in a specified index

Syntax

ADSKeyCount([<xTag>], <cIgnoredIndexFile>, [<nFilterOption>]) --> nKeyCount

Arguments

<xTag> Numeric order number OR index tag name. Default is current index.

<cIgnoredIndexFile> This parameter is not processed. In other Harbour RDDs, the second parameter to "ordKeyCount" takes a second argument to identify a particular Index File in cases where two files are open that contain orders with the same name. The ADS driver does not support this and will select the first order with the requested name. To stay consistent with other RDDs, therefore(),the second parameter is reserved and the <nFilterOption> is passed as a third parameter.

<nFilterOption> Indicates if filters and/or scopes are to be respected if set.

ADS_RESPECTFILTERS	Respect filters and scopes
ADS_IGNOREFILTERS	Ignore filters and scopes
ADS_RESPECTSCOPES	Respect scopes only

Returns

<nKeyCount> The number of keys within the current index.

Description

See ACE.HLP for full details about the Advantage Database Server.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSKeyNo()
ADSGetRelKeyPos()

ADSKeyNo()

Get the logical key number of the current record in the given index

Syntax

ADSKeyNo([<xTag>], [<nFilterOption>]) --> nKeyNo

Arguments

<xTag> Numeric order number OR index tag name. Default is current index.

<nFilterOption> Indicates if filters and/or scopes are to be respected if set.

ADS_RESPECTFILTERS	
ADS_IGNOREFILTERS	
ADS_RESPECTSCOPES	

Returns

<nKeyNo> The logical key number of the current record in the given index.

Description

See ACE.HLP for full details about the Advantage Database
Wrapper for AdsGetKeyNum.
This function may be slow on a large database with ADS_RESPECTFILTERS set
because it walks through the keys to get the current position. Compare to
ADSGetRelKeyPos().

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSKeyCount()
ADSGetRelKeyPos()

ADSLocking()

Turns on/off the Advantage proprietary locking mode

Syntax

ADSLocking(<lMode>) --> lPriorSetting

Arguments

<lMode> .T. to use the Advantage proprietary locking mode (this is the default setting if a remote server is used) or pass .F. to use "compatibility" locking.

Returns

<lPriorSetting> .T. if prior setting was for the proprietary mode.

Description

See ACE.HLP for full details about the Advantage Database Server. The Advantage Database Server has a fast Proprietary locking mode that is more efficient than traditional network locking. It is only available when using the remote server (not the local server).

If a file is opened in the proprietary mode, other applications cannot open it in a "write" mode. So if non-Advantage applications need concurrent access to the data files, use the Compatibility locking mode by calling ADSLocking(.F.).

ADSLocking() is a Get/Set function for the locking mode. It affects files at the time they are opened. So when a data file is opened, the current setting is used for that file until it is closed. Different files can have different locking modes by changing the setting before opening a second file.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSRightsCheck()

ADSRightsCheck()

Sets the "rights checking" setting for opening files

Syntax

ADSRightsCheck(<lMode>) --> lPriorSetting

Arguments

<lMode> .T. to check rights upon opening data files (the default), or .F. to ignore rights

Returns

Description

See ACE.HLP for full details about the Advantage Database Server. ADSRightsCheck() is a Get/Set function for the "rights checking" mode. If the setting is .T. when a file is opened, then the Advantage Database Server will use the rights of the connected user when opening the file. If the user does not have rights to the directory or server, then the open call will fail.

If the setting is .F., then the ADS will ignore the connected user's rights and open the file regardless. This lets you allow only Advantage-based applications to access specific data.

Status

Ready

Compliance

Harbour extension

Platforms

Windows 32-bit

Files

Library is RddAds Header is ads.ch

See Also:

ADSLocking()

HB_ZIPFILE()

Create a zip file

Syntax

HB_ZIPFILE(<cFile> , <cFileToCompress> | <aFiles>, <nLevel> , <bBlock>,<lOverWrite> ,<cPassword>,<lWithPath>,<lWithDrive>) ---> lCompress

Arguments

<cFile> Name of the zip file

<cFileToCompress> Name of a file to Compress, Drive and/or path can be used

<aFiles> An array containing files to compress, Drive and/or path can be used

<nLevel> Compression level ranging from 0 to 9

<bBlock> Code block to execute while compressing

<lOverWrite> Toggle to overwite the file if exists

<cPassword> Password to encrypt the files

<lWithPath> Toggle to store the path or not

<lWithDrive> Toggle to store the Drive letter and path or not

Returns

<lCompress> .t. if file was create, otherwise .f.

Description

This function creates a zip file named <cFile>. If the extension is ommited, .ZIP will be assumed. If the second parameter is a character string, this file will be added to the zip file. If the second parameter is an array, all file names contained in <aFiles> will be compressed.

If <nLevel> is used, it detemines the compression type where 0 means no compression and 9 means best compression.

If <bBlock> is used, every time the file is opened to compress it will evaluate bBlock. Parameters of bBlock are cFile and nPos.

If <lOverWrite> is used , it toggles to overwrite or not the existing file. Default is to overwrite the file,otherwise if <lOverWrite> is false the new files are added to the <cFile>.

If <cPassword> is used, all files that are added to the archive are encrypted with the password.

If <lWithPath> is used, it tells thats the path should also be stored with the file name. Default is false.

If <lWithDrive> is used, it tells thats the Drive and path should also be stored with the file name. Default is false.

```
FUNCTION MAIN()

IF HB_ZIPFILE( "TEST.ZIP","TEST.PRG")
   qout("File was successly create")
ENDIF

IF HB_ZIPFILE( "TEST1.ZIP",{"TEST.PRG","c:\windows\win.ini"})
   qout("File was successly create")
ENDIF

IF HB_ZIPFILE( "TEST2.ZIP",{"TEST.PRG","c:\windows\win.ini"},8,{|nPos,cFile|,qout(cFile)}
   qout("File was successly create")
ENDIF

aFiles := {"TEST.PRG","c:\windows\win.ini"}
nLen   := Len(afiles)
aGauge := GaugeNew( 5, 5, 7,40 , "W/B", "W+/B" ,'²')
```

```
    GaugeDisplay( aGauge )
    Hb_ZIPFILE('test33.zip',aFiles,8,{|cFile,nPos| GaugeUpdate(aGauge,nPos/nLen)},,'hello')
    Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_UNZIPFILE()

Unzip a compressed file

Syntax

HB_UNZIPFILE(<cFile> , <bBlock> , <lWithPath>) ,<cPassWord>,<cPath>, [<cFile>|<aFile>] <---> lCompress

Arguments

<cFile> Name of the zip file

<bBlock> Code block to execute while compressing

<lWithPath> Toggle to create directory if needed

<cPassWord> Password to use to extract files

<cPath> Path to extract the files.

<cFile>|<aFiles> An file or an Array of files to extract

Returns

<lCompress> .t. if all file was successfuly restored, otherwise .f.

Description

This function restores all files contained inside the <cFile>. If the extension is ommited, .ZIP will be assumed. If a file already exists, it wlll be overwriten.

If <bBlock> is used, every time the file is opened to compress it will evaluate bBlock. Parameters of bBlock are cFile and nPos.

The <cPath> is an obrogatory parameter. Set to ".\" to extract to the current dir

Examples

```
FUNCTION MAIN()

IF HB_UNZIPFILE( "TEST.ZIP")
   qout("File was successly create")
ENDIF

IF HB_ZIPFILE( "TEST2.ZIP",{|cFile|,qout(cFile)})
   qout("File was successly create")
ENDIF

Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_GETUNZIPFILE()

Gets the number of files that are in the zipfile

Syntax

HB_GETUNZIPFILE(<cFile>) ---> nNumber

Arguments

<cFile> Name of the zip file

Returns

<nNumber> The number of files contained inside the zipfile

Description

This function returns the number of files that is stored in the zipfile. The purpose for this function is to use in conjuntion with the HB_UNZIPFILE() function, so you can use returned result in the code block. See example below.

Examples

```
FUNCTION MAIN()
Local nFiles :=HB_GETUNZIPFILE('test.zip')

if nFiles >0
   ? "This files Contains ",nfiles
endif

Return Nil

Here is an example of How to use HB_GETUNZIPFILE() in conjunction
with HB_UNZIPFILE()

Function Main()
Local aGauge,nLen

aGauge := GaugeNew( 5, 5, 7,40 , "W/B", "W+/B" ,'²')
GaugeDisplay( aGauge )
nLen := HB_GETUNZIPFILE('test22')
hb_UNZIPFILE('test22',{|cFile,nPos| GaugeUpdate(aGauge,nPos/nLen),qout(cFile)},.t.)

Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_ZIPFILEBYTDSPAN()

Create a zip file

Syntax

HB_ZIPFILEBYTDSPAN()(<cFile> , <cFileToCompress> | <aFiles>, <nLevel> , <bBlock>,<lOverWrite> ,<cPassword>,<iSize>,<lWithPath>,<lWithDrive>) ---> lCompress

Arguments

<cFile> Name of the zip file

<cFileToCompress> Name of a file to Compress, Drive and/or path can be used

<aFiles> An array containing files to compress, Drive and/or path can be used

<nLevel> Compression level ranging from 0 to 9

<bBlock> Code block to execute while compressing

<lOverWrite> Toggle to overwite the file if exists

<cPassword> Password to encrypt the files

<iSize> Size of the archive, in bytes,default is 1457664 bytes

<lWithPath> Toggle to store the path or not

<lWithDrive> Toggle to store the Drive letter and path or not

Returns

<lCompress> .t. if file was create, otherwise .f.

Description

This function creates a zip file named <cFile>. If the extension is ommited, .ZIP will be assumed. If the second parameter is a character string, this file will be added to the zip file. If the second parameter is an array, all file names contained in <aFiles> will be compressed.

If <nLevel> is used, it detemines the compression type where 0 means no compression and 9 means best compression.

If <bBlock> is used, every time the file is opened to compress it will evaluate bBlock. Parameters of bBlock are cFile and nPos.

If <lOverWrite> is used , it toggles to overwrite or not the existing file. Default is to overwrite the file,otherwise if <lOverWrite> is false the new files are added to the <cFile>.

If <lWithPath> is used, it tells thats the path should also be stored with the file name. Default is false.

If <lWithDrive> is used, it tells thats the Drive and path should also be stored with the file name. Default is false.

```
FUNCTION MAIN()

IF HB_ZIPFILEBYTDSPAN( "TEST.ZIP","TEST.PRG")
   qout("File was successly create")
ENDIF

IF ZIPFILEBYTDSPAN( "TEST1.ZIP",{"TEST.PRG","c:\windows\win.ini"})
   qout("File was successly create")
ENDIF

IF HB_ZIPFILEBYTDSPAN( "TEST2.ZIP",{"TEST.PRG","c:\windows\win.ini"},8,{|nPos,cFile|,qout
   qout("File was successly create")
ENDIF

aFiles := {"TEST.PRG","c:\windows\win.ini"}
nLen   := Len(afiles)
aGauge := GaugeNew( 5, 5, 7,40 , "W/B", "W+/B" ,'²')
GaugeDisplay( aGauge )
```

```
    HB_ZIPFILEBYTDSPAN('test33.zip',aFiles,8,{|cFile,nPos| GaugeUpdate(aGauge,nPos/nLen)},,'h
    Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_ZIPFILEBYPKSPAN()

Create a zip file on removable media

Syntax

HB_ZIPFILEBYPKSPAN(<cFile> , <cFileToCompress> | <aFiles>, <nLevel> , <bBlock>,<lOverWrite> ,<cPassword>,<lWithPath>,<lWithDrive>) ---> lCompress

Arguments

<cFile> Name of the zip file

<cFileToCompress> Name of a file to Compress, Drive and/or path can be used

<aFiles> An array containing files to compress, Drive and/or path can be used

<nLevel> Compression level ranging from 0 to 9

<bBlock> Code block to execute while compressing

<lOverWrite> Toggle to overwite the file if exists

<cPassword> Password to encrypt the files

<lWithPath> Toggle to store the path or not

<lWithDrive> Toggle to store the Drive letter and path or not

Returns

<lCompress> .t. if file was create, otherwise .f.

Description

This function creates a zip file named <cFile>. If the extension is ommited, .ZIP will be assumed. If the second parameter is a character string, this file will be added to the zip file. If the second parameter is an array, all file names contained in <aFiles> will be compressed.Also , the use of this function is for creating backup in removable media like an floppy drive/zip drive.

If <nLevel> is used, it detemines the compression type where 0 means no compression and 9 means best compression.

If <bBlock> is used, every time the file is opened to compress it will evaluate bBlock. Parameters of bBlock are cFile and nPos.

If <lOverWrite> is used , it toggles to overwrite or not the existing file. Default is to overwrite the file,otherwise if <lOverWrite> is false the new files are added to the <cFile>.

If <cPassword> is used, all files that are added to the archive are encrypted with the password.

If <lWithPath> is used, it tells thats the path should also be stored with the file name. Default is false.

If <lWithDrive> is used, it tells thats the Drive and path should also be stored with the file name. Default is false.

Before calling this function, Set an Changedisk codeblock by calling the HB_SETDISKZIP().

Examples

```
FUNCTION MAIN()

IF HB_ZIPFILE( "a:\TEST.ZIP","TEST.PRG")
   qout("File was successly create")
ENDIF

IF HB_ZIPFILEBYPKSPAN( "a:\TEST1.ZIP",{"TEST.PRG","c:\windows\win.ini"})
   qout("File was successly create")
ENDIF

IF HB_ZIPFILEBYPKSPAN( "TEST2.ZIP",{"TEST.PRG","c:\windows\win.ini"},8,{|nPos,cFile|,qout
   qout("File was successly create")
ENDIF

aFiles := {"TEST.PRG","c:\windows\win.ini"}
nLen   := Len(afiles)
aGauge := GaugeNew( 5, 5, 7,40 , "W/B", "W+/B" ,'²')
GaugeDisplay( aGauge )
Hb_ZIPFILEBYPKSPAN('f:\test33.zip',aFiles,8,{|cFile,nPos| GaugeUpdate(aGauge,nPos/nLen)},
// assuming f:\ as an Zip Drive
Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_SETDISKZIP()

Set an codeblock for disk changes

Syntax

```
HB_SETDISKZIP(<bBlock>)
```

Arguments

<bBlock> an Code block that contains an function that will be performed when the need of changing disk are need.

Returns

Description

This function will set an codeblock that will be evaluated every time that an changedisk event is neccessary.

Set this function before opening archives that are in removable media. This block will be released, when the caller finish it job.

Examples

```
HB_SETDISKZIP({|x| alert('Please insert disk no "+str(x,3))})
```

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_ZIPDELETEFILES()

Delete files from an zip archive

Syntax

HB_ZIPDELETEFILES(<cFile>,<cFiletoDelete>|<aFiles>) --> <lDeleted>

Arguments

<cFile> The name of the zip files from where the files will be deleted

<cFiletoDelete> An File to be removed

<aFiles> An Array of Files to Be Removed

Returns

<lDeleted> If the files are deleted , it will return .T., otherwise it will return .f. for the Follow cases: Spanned Archives, the file could not be found in the zip file.

Description

This function removes files from an Zip archive.

Examples

```
? "has the file zipnew.i been deleted ",if(HB_ZIPDELETEFILES('\test23.zip','zipnew.i'),"Y
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_ZIPTESTPK()

Test pkSpaned zip files

Syntax

HB_ZIPTESTPK(<cFile>) --> <nReturnCode>

Arguments

<cFile> File to be tested.

Returns

<nReturn> An code that tell if the current disk is the Last of an pkspanded disk set.

Description

This function tests if the disk inserted is the last disk of an backup set or not. It will return the follow return code when an error is found

Error code	Meaning
114	Incorrect Disk
103	No Call back was set with HB_SETDISKZIP()

Call this function to determine if the disk inserted is the correct one before any other function.

Examples

```
if (HB_ZIPTESTPK('a:\test22.zip')==114)
    ? "invalid Disk"
endif
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_GETZIPCOMMENT()

Return the comment of an zip file

Syntax

```
HB_GETZIPCOMMENT(<szFile>) --> <szComment>
```

Arguments

<szFile> File to get the comment from

Returns

<szComment> The comment that was stored in <szFile>

Description

This function recieve an valid zip file name as parameter, and return the global comment stored within.

Examples

```
? "The comment of tes.zip",HB_GETZIPCOMMENT('tes.zip')
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_SETZIPCOMMENT()

Set an Zip archive Comment

Syntax

```
HB_SETZIPCOMMENT(<cComment>) -->Nil
```

Arguments

<cComment> Comment to add to the zip archive

Returns

<NIL> this function always return NIL

Description

This function stored an global comment to an zip archive. It should be called before any of the Compression Function.

Examples

```
HB_SETZIPCOMMENT("This is an Test")
hb_zipfile('test.zip',{"\windows\ios.ini","\windows\win.ini"})
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_SETBUFFER()

Syntax

HB_SETBUFFER([<nWriteBuffer>],[<nExtractBuffer>],[<nReadBuffer>]) -->Nil

Arguments

<nWriteBuffer>	The size of the write buffer.
<nExtractBuffer>	The size of the extract buffer.
<nReadBuffer>	The size of the read buffer.

Returns

<NIL>	This function always return NIL.

Description

This function set the size of the internal buffers for read/write/extract operation

If the size of the buffer is smaller then the default , the function will automaticaly use the default values,which is 65535/16384/32768.

This function be called before any of the Compression/decompression Function.

Examples

```
HB_SETBUFFER(100000,115214,65242)
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

HB_GETUNRARFILE()

Gets the number of files that are in the rar file

Syntax

HB_GETUNRARFILE(<cFile>) ---> nNumber

Arguments

<cFile> Name of the rar file

Returns

<nNumber> The number of files contained inside the rarfile

Description

This function returns the number of files that is stored in the rar file.

Examples

```
FUNCTION MAIN()
Local nFiles :=HB_GETUNZIPFILE('test.zip')

if nFiles >0
   ? "This files Contains ",nfiles
endif

Return Nil
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

Files

Library is zlib.lib and zlib_bor.lib For Borland Compilers Library is zlib.lib zlib_ms.lib for MSVC compilers

COMPRESSSTRING()

Compress an string

Syntax

CompressString(@<cString>,@<nSize>)

Arguments

<cString>	Buffer to compress, must be passed by reference.
<nSize>	The Lenght of the compression String

Returns

Description

This function compress an string buffer and return the compression function and the size of this compression function. This size is required for uncompression the string

Examples

```
Local cTest:= memoread('tzipfile.prg')
? 'String before compressing',cTest

COMPRESSSTRING(@cTest,@nSize)
? "Size",nSize
? " "
? " "
? 'String after compressing',cTest
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

See Also:

UNCOMPRESSSTRING()

UNCOMPRESSSTRING()

Uncompress an String

Syntax

UNCOMPRESSSTRING(<cBuffer>,<nSize>) ---> <cUncompressBuffer>

Arguments

<cBuffer>	The compressing string to uncompress
<nSize>	The saved size returned by compressionstring

Returns

<cUncompressBuffer>	The String uncompressed

Description

```
cOrig:=UNCOMPRESSSTRING(cTest,nSize)
?'uncompressed String',cOrig
```

Status

Ready

Compliance

This function is a Harbour extension

Platforms

All

See Also:

COMPRESSSTRING()

ADDASCII()
Add an integer value to an ascii value of a string

Syntax

ADDASCII (<[@]cString>, <nValue>, [<nPosition>], [<lCarryOver>]) --> cString

Arguments

<[@]cString> is the string that should be edited <nValue> is a integer value that should be added to the ASCII value of the character at the <nPosition>th position [<nPosition>] is the position of the character that should be edited. If not supplied, the last character of <[@]cString> is edited. [<lCarryOver>] NEW: is set to .T. if the substring from position 1 to position <nPosition> should be treated as an integer written to the base 256. Thus, the addition of <nValue> can affect to whole substring (see EXAMPLES). Default is .F., the original behaviour of this function.

Returns

CSETREF() function. The string must then be passed by reference [@].

Description

ADDASCII() can be used to add or subtract integer values from ASCII values in a string. The new <lCarryOver> parameter allows to treat a string as an integer written to the base 256. Since <nValue> is limited to a signed long, only substrings 4 characters long can be affected by one ADDASCII() call. If the length of <[@]cString> is smaller than <nPosition>, the string remains unchanged. The same happens, if uninterpretable parameters are passed to this function.

Examples

```
// Add 32 to the ASCII value of the character at the last position
// in the string

? addascii ("SmithH", 32)  --> "Smith"
```

Tests

```
addascii ("0000", 1, 1) == "1000"
addascii ("0000", 1) == "0001"
addascii ("AAAA", -255, 1) == "BAAA"
addascii ("AAAA", -255) == "AAAB"
addascii ("AAAA", 1, 2, .T.) == "ABAA"
addascii ("AAAA", 257, 2, .T.) == "BBAA"
addascii ("AAAA", 257, 2, .F.) == "ABAA"
addascii ("AAAA", 258,, .T.) == "AABC"
addascii ("ABBA", -257, 3, .T.) == "AAAA"
```

Status

Ready

Compliance

ADDASCII() is compatible with CT3's ADDASCII(). A new, 4th, parameter has been added who defaults to the original behaviour if omitted.

Platforms

All

Files

Source is addascii.c, library is ct3.

See Also:

CSETREF()

ATADJUST()

Adjusts a sequence within a string to a specified position

Syntax

ATADJUST (<cStringToMatch>, <cString>, <nAdjustPosition>,
[<nCounter>], [<nIgnore>],
[<nFillChar|cFillChar>]) -> cString

Arguments

<cStringToMatch> is the sequence to be adjusted within <cString>
<cString> is the string that contains <cStringToMatch>
<nAdjustPosition> specifies the position to that <cStringToMatch> will be adjusted [<nCounter>] specifies which occurence of <cStringToMatch> in <cString> is to be adjusted Default: last occurence [<nIgnore>] specifies how many characters should be omitted in the scan
[<nFillChar|cFillChar>] specifies the character that is used for the adjustment

Returns

Description

<TODO: add a description, some examples and tests here>

Status

Ready

Compliance

ATADJUST() works like CT3's ATADJUST()

Platforms

All

Files

Source is atadjust.c, library is ct3.

See Also:

SETATLIKE()
CSETATMUPA()

AFTERATNUM()

Returns string portion after nth occurence of substring

Syntax

AFTERATNUM (<cStringToMatch>, <cString>, [<nCounter>], [<nIgnore>]) --> cRestString

Arguments

<cStringToMatch> is the substring scanned for <cString> is the scanned string [<nCounter>] determines how many occurences are of <cStringToMatch> in <cString> are searched Default: search last occurence [<nIgnore>] determines how many character from the start should be ignored in the search Default: 0

Returns

<cRestString> the portion of <cString> after the <nCounter>th occurence of <cStringToMatch> in <cString> If such a rest does not exist, an empty string is returned.

Description

This function scans <cString> for <cStringToMatch>. After the <nCounter>th match (or the last one, depending on the value of <nCounter>) has been found, the portion of <cString> after that match will be returned. If there aren't enough matches or the last match is identical to the end of <cString>, an empty string will be returned. After a match has been found, the function continues to scan after that match if the CSETATMUPA() switch is turned off, with the second character of the matched substring otherwise. The function will also consider the settings of SETATLIKE().

Examples

```
? AFTERATNUM ("!", "What is the answer ? 4 ! 5 !") -> ""
? AFTERATNUM ("!", "What is the answer ? 4 ! 5 ?") -> " 5 ?"
<TODO: add some examples here with csetatmupa() and setatlike()>
```

Tests

```
AFTERATNUM ("..", "..This..is..a..test!") == "test!"
AFTERATNUM ("..", "..This..is..a..test!", 2) == "is..a..test!"
AFTERATNUM ("..", "..This..is..a..test!", 2, 2) == "a..test!"
```

Status

Ready

Compliance

AFTERATNUM() is compatible with CT3's AFTERATNUM().

Platforms

All

Files

Source is atnum.c, library is libct.

See Also:

ATNUM()
BEFORATNUM()
CSETATMUPA()
SETATLIKE()

BEFORATNUM()

Returns string portion before nth occurence of substring

Syntax

BEFORATNUM (<cStringToMatch>, <cString>, [<nCounter>], [<nIgnore>]) --> cRestString

Arguments

<cStringToMatch> is the substring scanned for <cString> is the scanned string [<nCounter>] determines how many occurences are of <cStringToMatch> in <cString> are searched Default: search last occurence [<nIgnore>] determines how many character from the start should be ignored in the search Default: 0

Returns

<cRestString> the portion of <cString> before the <nCounter>th occurence of <cStringToMatch> in <cString> If such a string does not exist, an empty string is returned.

Description

This function scans <cString> for <cStringToMatch>. After the <nCounter>th match (or the last one, depending on the value of <nCounter>) has been found, the portion of <cString> before that match will be returned. If there aren't enough matches or the last match is identical to the start of <cString> (i.e. the last match is the first match), an empty string will be returned. After a match has been found, the function continues to scan after that match if the CSETATMUPA() switch is turned off, with the second character of the matched substring otherwise. The function will also consider the settings of SETATLIKE().

Examples

```
? BEFORATNUM ("!", "What is the answer ? 4 ! 5 !") -> "What is the answer ? 4 ! 5 "
? BEFORATNUM ("!", "What is the answer ? 4 ! 5 ?") -> "What is the answer ? 4 "
<TODO: add some examples here with csetatmupa() and setatlike()>
```

Tests

```
BEFORATNUM ("..", "..This..is..a..test!") == "..This..is..a"
BEFORATNUM ("..", "..This..is..a..test!", 2) == "..This"
BEFORATNUM ("..", "..This..is..a..test!", 2, 2) == "..This..is"
```

Status

Ready

Compliance

BEFORATNUM() is compatible with CT3's BEFORATNUM().

Platforms

All

Files

Source is atnum.c, library is ct3.

See Also:

ARRAY()

ATNUM()

Returns the start position of the nth occurence of a substring in a string

Syntax

```
ATNUM (<cStringToMatch>, <cString>, [<nCounter>],
[<nIgnore>] ) --> nPosition
```

Arguments

<cStringToMatch> is the substring scanned for <cString> is the scanned string [<nCounter>] determines how many occurences are of <cStringToMatch> in <cString> are searched Default: search last occurence [<nIgnore>] determines how many character from the start should be ignored in the search Default: 0

Returns

<nPosition> the position of the <nCounter>th occurence of <cStringToMatch> in <cString>. If such an occurence does not exist, 0 is returned.

Description

This function scans <cString> for <cStringToMatch>. After the <nCounter>th match (or the last one, depending on the value of <nCounter>) has been found, the position of that match will be returned. If there aren't enough matches or there is no last match, 0 will be returned. After a match has been found, the function continues to scan after that match if the CSETATMUPA() switch is turned off, with the second character of the matched substring otherwise. The function will also consider the settings of SETATLIKE().

Examples

```
? ATNUM ("!", "What is the answer ? 4 ! 5 !") -> 28
? ATNUM ("!", "What is the answer ? 4 ! 5 ?") -> 24
<TODO: add some examples here with csetatmupa() and setatlike()>
```

Tests

```
ATNUM ("..", "..This..is..a..test!") == 14
ATNUM ("..", "..This..is..a..test!", 2) == 7
ATNUM ("..", "..This..is..a..test!", 2, 2) == 11
```

Status

Ready

Compliance

ATNUM() is compatible with CT3's ATNUM().

Platforms

All

Files

Source is atnum.c, library is libct.

See Also:

ARRAY()

CSETREF()

Determine return value of reference sensitive CT3 string functions

Syntax

CSETREF ([<lNewSwitch>]) -> lOldSwitch

Arguments

[<lNewSwitch>] .T. -> suppress return value .F. -> do not suppress return value

Returns

switch

Description

Within the CT3 functions, the following functions do not change the length of a string passed as parameter while transforming this string:

```
ADDASCII()   BLANK()       CHARADD()  CHARAND()    CHARMIRR()    CHARNOT()
CHAROR()     CHARRELREP()  CHARREPL()  CHARSORT()   CHARSWAP()    CHARXOR()
CRYPT()      JUSTLEFT()    JUSTRIGHT()  POSCHAR()    POSREPL()     RANGEREPL()
REPLALL()    REPLLEFT()    REPLRIGHT()  TOKENLOWER() TOKENUPPER()  WORDREPL()
WORDSWAP()
```

Thus, these functions allow to pass the string by reference [@] to the function so that it may not be necessary to return the transformed string. By calling CSETREF (.T.), the above mentioned functions return the value .F. instead of the transformed string if the string is passed by reference to the function. The switch is turned off (.F.) by default.

Status

Ready
This function is fully CT3 compatible.

Platforms

All

Files

Source is ctstr.c, library is ct3.

See Also:

WORDSWAP()

CSETATMUPA()

Determine "multi-pass" behaviour in some string functions

Syntax

CSETATMUPA ([<lNewSwitch>]) -> lOldSwitch

Arguments

[<lNewSwitch>] .T. -> turn "multi-pass" on .F. -> turn "multi-pass" off

Returns

switch

Description

CSETATMUPA determines how the following CT3 string functions

```
ATNUM()       AFTERATNUM()  BEFORATNUM()  ATREPL()      NUMAT()
ATADJUST()  WORDTOCHAR()  WORDREPL()
```

perform their work. See the respective function documentation for a further description how the switch influences these functions.

Status

Ready
This function is fully CT3 compatible.

Platforms

All

Files

Source is ctstr.c, library is ct3.

See Also:

ARRAY()

SETATLIKE()

Determine scan behaviour in some string functions

Syntax

SETATLIKE ([<nMode>] [, <[@]cWildcard>]) --> nOldMode

Arguments

[<nMode>] CT_SETATLIKE_EXACT -> characters are compared exactly CT_SETATLIKE_WILDCARD -> characters are compared using a wildcard character The default value is CT_SETATLIKE_EXACT. [<[@]cWildcard>] determines the character that is subsequently used as a wildcard character for substring scanning. The default value is "?". NEW: If this parameter is passed by reference [@], the current wildcard character is stored in <cWildcard>.

Returns

Description

In the following CT3 functions, strings are compared on a character base:

ATADJUST() ATNUM() AFTERATNUM() BEFOREATNUM() ATREPL() NUMAT() STRDIFF()

With the SETATLIKE function, one can determine when characters are considered to match within these functions. If CT_SETATLIKE_WILDCARD is set (e.g. "?"), then "?" matches every other character.

<nMode> can be one of the following values that are defined in ct.ch

Definition	Value
CT_SETATLIKE_EXACT	0
CT_SETATLIKE_WILDCARD	1

Status

Ready

This function is fully CT3 compatible, but allows to pass the second parameter by reference so that the current wildcard character can be determined.

Platforms

All

Files

Source is ctstr.c, header is ct.ch, library is ct3.

CHAREVEN()

Returns the characters on the even positions in a string

Syntax

```
CHAREVEN (<cString>) --> cEvenString
```

Arguments

<cString>	processed string

Returns

<cEvenString>	a string containing all character from even positions in <cString>

Description

The CHAREVEN() function looks for the characters on the even positions in a given string, collects them and returns them as a string.

Examples

```
? CHAREVEN (" H E L L O !") -> "HELLO!"
```

Tests

```
CHAREVEN (" 1 2 3 4 5") == "12345"
CHAREVEN (" 1 2 3 4 ") == "1234"
CHAREVEN (" ") == ""
```

Status

Ready

Compliance

CHAREVEN() is compatible with CT3's CHAREVEN().

Platforms

All

Files

Source is charevod.c, library is ct3.

See Also:

ARRAY()

CHARODD()

Returns the characters on the odd positions in a string

Syntax

```
CHARODD (<cString>) --> cOddString
```

Arguments

<cString>	processed string

Returns

<cOddString>	a string containing all character from odd positions in <cString>

Description

The CHARODD() function looks for the characters on the odd positions in a given string, collects them and returns them as a string.

Examples

```
? CHARODD ("H E L L O ! ") -> "HELLO!"
```

Tests

```
CHARODD ("1A2B3C4D5E") == "12345"
CHARODD ("1A2B3C4D5")  == "12345"
```

Status

Ready

Compliance

CHARODD() is compatible with CT3's CHARODD().

Platforms

All

Files

Source is charevod.c, library is ct3.

See Also:

ARRAY()

CHARADD()

Adds corresponding ASCII value of two strings

Syntax

CHARADD (<[@]cString1>, <cString2>) --> cAddString

Arguments

<[@]cString1> first string <cString2> second string

Returns

<cAddString> string with added ASCII values

Description

The CHARADD() function constructs a new string from the two strings passed as parameters. To do this, it adds the ASCII values of the corresponding characters of both strings and places a character in the resulting string whose ASCII value equals to that sum (modulo 256). If the first string is passed by reference, the resulting string is stored in <cString1>, too. By setting the CSETREF()-switch to .T., the return value can be omitted. If <cString2> is shorter than <cString1> and the last character of <cString2> has been processed, the function restarts with the first character of <cString2>.

Examples

```
? charadd ("012345678", chr(1)) --> "123456789"
? charadd ("123456789", chr(255)) --> "012345678"
? charadd ("0000", chr(0)+chr(1)+chr(2)+chr(3)) --> "0123"
```

Tests

```
charadd ("012345678", chr(1)) == "123456789"
charadd ("012345678", chr(1)+chr(2)) == "133557799"
charadd ("123456789", chr(255)) == "012345678"
charadd ("123456789", chr(255)+chr(254)) == "002244668"
```

Status

Ready

Compliance

CHARADD() is compatible with CT3's CHARADD().

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARSUB()

Subtracts corresponding ASCII value of two strings

Syntax

CHARSUB (<[@]cString1>, <cString2>) --> cSubString

Arguments

<[@]cString1> first string <cString2> second string

Returns

<cSubString>	string with subtracted ASCII values

Description

The CHARSUB() function constructs a new string from the two strings passed as parameters. To do this, it subtracts the ASCII values of the corresponding characters of both strings and places a character in the resulting string whose ASCII value equals to that difference (modulo 256). If the first string is passed by reference, the resulting string is stored in <cString1>, too. By setting the CSETREF()-switch to .T., the return value can be omitted. If <cString2> is shorter than <cString1> and the last character of <cString2> has been processed, the function restarts with the first character of <cString2>.

Examples

```
? charsub ("012345678", chr(1)) --> "/01234567"
? charsub ("123456789", chr(255)) --> "23456789:"
? charsub ("9999", chr(0)+chr(1)+chr(2)+chr(3)) --> "9876"
```

Tests

```
charsub ("123456789", chr(1)) == "012345678"
charsub ("123456789", chr(1)+chr(2)) == "002244668"
charsub ("012345678", chr(255)) == "123456789"
charsub ("012345678", chr(255)+chr(254)) == "133557799"
```

Status

Ready

Compliance

CHARSUB() is a new function that is only available in Harbour's CT3 lib.

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARAND()

Combine corresponding ASCII value of two strings with bitwise AND

Syntax

CHARAND (<[@]cString1>, <cString2>) --> cAndString

Arguments

<[@]cString1> first string <cString2> second string

Returns

<cAndString> string with bitwise AND combined ASCII values

Description

The CHARAND() function constructs a new string from the two strings passed as parameters. To do this, it combines the ASCII values of the corresponding characters of both strings with a bitwise AND-operation and places a character in the resulting string whose ASCII value equals to the result of that operation. If the first string is passed by reference, the resulting string is stored in <cString1>, too. By setting the CSETREF()-switch to .T., the return value can be omitted. If <cString2> is shorter than <cString1> and the last character of <cString2> has been processed, the function restarts with the first character of <cString2>.

Examples

```
// clear the LSB
? charand ("012345678", chr(254)) --> "002244668"
? charand ("012345678", chr(254)+chr(252)) --> "002044648"
```

Tests

```
charand ("012345678", chr(254)) == "002244668"
charand ("012345678", chr(254)+chr(252)) == "002044648"
```

Status

Ready

Compliance

CHARAND() is compatible with CT3's CHARAND().

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARNOT()

Process each character in a string with bitwise NOT operation

Syntax

```
CHARNOT (<[@]cString>) --> cNotString
```

Arguments

<[@]cString>	string to be processed

Returns

<cNotString>	string with bitwise negated characters

Description

The CHARNOT() function constructs a new string from the string passed as parameter. To do this, it performs a bitwise NOT operation to the characters of the string and places a character in the resulting string whose ASCII value equals to the result of that operation. It can be easily seen that the resulting ASCII-value equals 255 minus input ASCII value. If the string is passed by reference, the resulting string is stored in <cString>, too. By setting the CSETREF()-switch to .T., the return value can be omitted.

Examples

```
? charnot (chr(85)+chr(128)+chr(170)+chr(1)) --> chr(170)+chr(127)+chr(85)+chr(254)
? charnot (charnot ("This is a test!")) --> "This is a test!"
```

Tests

```
charnot (chr(85)+chr(128)+chr(170)+chr(1)) == chr(170)+chr(127)+chr(85)+chr(254)
charnot (charnot ("This is a test!")) == "This is a test!"
```

Status

Ready

Compliance

CHARNOT() is compatible with CT3's CHARNOT().

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHAROR()

Combine corresponding ASCII value of two strings with bitwise OR

Syntax

CHAROR (<[@]cString1>, <cString2>) --> cOrString

Arguments

<[@]cString1> first string <cString2> second string

Returns

<cOrString> string with bitwise OR combined ASCII values

Description

The CHAROR() function constructs a new string from the two strings passed as parameters. To do this, it combines the ASCII values of the corresponding characters of both strings with a bitwise OR-operation and places a character in the resulting string whose ASCII value equals to the result of that operation. If the first string is passed by reference, the resulting string is stored in <cString1>, too. By setting the CSETREF()-switch to .T., the return value can be omitted. If <cString2> is shorter than <cString1> and the last character of <cString2> has been processed, the function restarts with the first character of <cString2>.

Examples

```
// set the LSB
? charor ("012345678", chr(1)) --> "113355779"
? charor ("012345678", chr(1)+chr(3)) --> "133357779"
```

Tests

```
charor ("012345678", chr(1)) == "113355779"
charor ("012345678", chr(1)+chr(3)) == "133357779"
```

Status

Ready

Compliance

CHAROR() is compatible with CT3's CHAROR().

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARXOR()

Combine corresponding ASCII value of two strings with bitwise XOR

Syntax

CHARXOR (<[@]cString1>, <cString2>) --> cXOrString

Arguments

<[@]cString1> first string <cString2> second string

Returns

<cXOrString> string with bitwise XOR combined ASCII values

Description

The CHARXOR() function constructs a new string from the two strings passed as parameters. To do this, it combines the ASCII values of the corresponding characters of both strings with a bitwise XOR-operation and places a character in the resulting string whose ASCII value equals to the result of that operation. If the first string is passed by reference, the resulting string is stored in <cString1>, too. By setting the CSETREF()-switch to .T., the return value can be omitted. If <cString2> is shorter than <cString1> and the last character of <cString2> has been processed, the function restarts with the first character of <cString2>.

Examples

```
// easy encryption
? charxor ("This is top secret !", "My Password") --> <encrypted sentence>
```

Tests

```
charxor (charxor ("This is top secret !", "My Password"), "My Password") == "This is top
```

Status

Ready

Compliance

CHARXOR() is compatible with CT3's CHARXOR().

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARSHL()

Process each character in a string with bitwise SHIFT LEFT operation

Syntax

```
CHARSHL (<[@]cString>, <nBitsToSHL> ) --> cSHLString
```

Arguments

<[@]cString> string to be processed <nBitsToSHL> number of bit positions to be shifted to the left

Returns

<cSHLString> string with bitwise shifted left characters

Description

The CHARSHL() function constructs a new string from the string passed as parameter. To do this, it performs a bitwise SHIFT LEFT (SHL) operation to the characters of the string and places a character in the resulting string whose ASCII value equals to the result of that operation. Be aware that bits shifted out of the byte are lost. If you need a bit rotation, use the CHARRLL() function instead. If the string is passed by reference, the resulting string is stored in <cString>, too. By setting the CSETREF()-switch to .T., the return value can be omitted.

Examples

```
? charshl (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3)
  --> chr(8)+chr(16)+chr(32)+chr(64)+chr(128)+chr(0)+chr(0)+chr(0)
```

Tests

```
charshl (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3) == chr(8)+chr(1
```

Status

Ready

Compliance

CHARSHL() is a new function that is only available in Harbour's CT3 lib.

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARSHR()

Process each character in a string with bitwise SHIFT RIGHT operation

Syntax

```
CHARSHR (<[@]cString>, <nBitsToSHR> ) --> cSHRString
```

Arguments

<[@]cString> string to be processed <nBitsToSHR> number of bit positions to be shifted to the right

Returns

<cSHRString> string with bitwise shifted right characters

Description

The CHARSHR() function constructs a new string from the string passed as parameter. To do this, it performs a bitwise SHIFT RIGHT (SHR) operation to the characters of the string and places a character in the resulting string whose ASCII value equals to the result of that operation. Be aware that bits shifted out of the byte are lost. If you need a bit rotation, use the CHARRLR() function instead. If the string is passed by reference, the resulting string is stored in <cString>, too. By setting the CSETREF()-switch to .T., the return value can be omitted.

Examples

```
? charshr (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3)
  --> chr(0)+chr(0)+chr(0)+chr(1)+chr(2)+chr(4)+chr(8)+chr(16)
```

Tests

```
charshr (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3) == chr(0)+chr(0
```

Status

Ready

Compliance

CHARSHR() is a new function that is only available in Harbour's CT3 lib.

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARRLL()

Process each character in a string with bitwise ROLL LEFT operation

Syntax

CHARRLL (<[@]cString>, <nBitsToRLL>) --> cRLLString

Arguments

<[@]cString> string to be processed <nBitsToRLL> number of bit positions to be rolled to the left

Returns

<cRLLString> string with bitwise rolled left characters

Description

The CHARRLL() function constructs a new string from the string passed as parameter. To do this, it performs a bitwise ROLL LEFT (RLL) operation to the characters of the string and places a character in the resulting string whose ASCII value equals to the result of that operation. Be aware that, in contrast to CHARSHL(), bits rolled out on the left are put in again on the right. If the string is passed by reference, the resulting string is stored in <cString>, too. By setting the CSETREF()-switch to .T., the return value can be omitted.

Examples

```
? charrll (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3)
  --> chr(8)+chr(16)+chr(32)+chr(64)+chr(128)+chr(1)+chr(2)+chr(4)
```

Tests

```
charrll (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3) == chr(8)+chr(1
```

Status

Ready

Compliance

CHARRLL() is a new function that is only available in Harbour's CT3 lib.

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

CHARRLR()

Process each character in a string with bitwise ROLL RIGHT operation

Syntax

CHARRLR (<[@]cString>, <nBitsToRLR>) --> cRLRString

Arguments

<[@]cString> string to be processed <nBitsToRLR> number of bit positions to be rolled to the right

Returns

<cRLRString> string with bitwise rolled right characters

Description

The CHARRLR() function constructs a new string from the string passed as parameter. To do this, it performs a bitwise ROLL RIGHT (RLR) operation to the characters of the string and places a character in the resulting string whose ASCII value equals to the result of that operation. Be aware that, in contrast to CHARSHR(), bits rolled out on the right are put in again on the left. If the string is passed by reference, the resulting string is stored in <cString>, too. By setting the CSETREF()-switch to .T., the return value can be omitted.

Examples

```
? charrlr (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3)
  --> chr(32)+chr(64)+chr(128)+chr(1)+chr(2)+chr(4)+chr(8)+chr(16)
```

Tests

```
charrlr (chr(1)+chr(2)+chr(4)+chr(8)+chr(16)+chr(32)+chr(64)+chr(128), 3) == chr(32)+chr(
```

Status

Ready

Compliance

CHARRLR() is a new function that is only available in Harbour's CT3 lib.

Platforms

All

Files

Source is charop.c, library is ct3.

See Also:

CSETREF()

ASCIISUM()

calculate the sum of the ASCII values of the characters in a string

Syntax

ASCIISUM (<cString>) --> nAsciiSum

Arguments

<cString>	the string to be processed

Returns

<nAsciiSum>	sum of the ASCII values in <cString>

Description

The ASCIISUM() function sums up the ASCII values of the characters in <cString>. Be aware that the function is not position sensitive, i.e. a change of position of a certain character in the string does not change the ascii sum.

Examples

```
? asciisum ("ABC")  -->  197
? asciisum ("ACB")  -->  197
```

Tests

```
asciisum (replicate ("A", 10000)) == 650000
asciisum ("0123456789") == 525
asciisum (nil) == 0
```

Status

Ready

Compliance

ASCIISUM() is compatible with CT3's ASCIISUM().

Platforms

All

Files

Source is asciisum.c, library is ct3.

See Also:

ARRAY()

ASCPOS()

ASCII value of a character at a certain position

Syntax

```
ASCPOS (<cString>, [<nPosition>]) --> nAsciiValue
```

Arguments

<cString> is the processed string [<nPosition>] is an optional position within <cString> Default: last position in <cString>

Returns

<nAsciiValue> the ASCII value of the character at the specified position

Description

The ASCPOS() function returns the ASCII value of the character that can be found at the position <nPosition> in <cString>. If <nPosition> is larger than the length of <cString>, 0 is returned.

Examples

```
? ascpos ("0123456789") --> 57
? ascpos ("0123456789",1) --> 48
```

Tests

```
ascpos ("0123456789") == 57
ascpos ("0123456789",1) == 48
ascpos ("0123456789",11) == 0  // <nPosition> to large !
```

Status

Ready

Compliance

ASCPOS() is compatible with CT3's ASCPOS().

Platforms

All

Files

Source is asciisum.c, library is libct.

See Also:

VALPOS()

VALPOS()

Numerical value of a character at a certain position

Syntax

```
VALPOS (<cString>, [<nPosition>]) --> nDigitValue
```

Arguments

<cString> is the processed string [<nPosition>] is an optional position within <cString> Default: last position in <cString>

Returns

<nDigitValue> the numerical value of the character at the specified position

Description

The VALPOS() function returns the numerical value of the character that can be found at the position <nPosition> in <cString>. If no digit can be found at this position or if <nPosition> is larger than the length of <cString>, 0 is returned.

Examples

```
? valpos ("1234x56789") --> 9
? valpos ("1234x56789",1) --> 1
```

Tests

```
valpos ("1234x56789") == 9
valpos ("1234x56789",1) == 1
valpos ("1234x56789",11) == 0  // <nPosition> to large !
valpos ("1234x56789",5) == 0   // "x" is not a digit !
```

Status

Ready

Compliance

VALPOS() is compatible with CT3's VALPOS().

Platforms

All

Files

Source is asciisum.c, library is libct.

See Also:

ASCPOS()

ATREPL()

Search and replace sequences in a string

Syntax

ATREPL (<cStringToMatch>, <cString>, <cReplacement>, [<nCounter>], [<lMode>], [<nIgnore>]) --> cString

Arguments

<cStringToMatch> is the substring searched for in <cString> <cString> is the processed string <cReplacement> is the replacement for sequences found [<nCounter>] specifies the number of replacements Default: last occurence [<lMode>] if set to .T., only the <nCounter>th sequence of <cStringToMatch> will be replaced, else all sequences will be replaced. Default: .F. [<nIgnore>]) specifies how many characters in <cString> from the beginning should be ignored by the function Default: 0

Returns

Description

The ATREPL() function searches and replaces sequences in a string. First, the function ignores the first <nIgnore> characters of <cString>. Then, if <lMode> is set to .T., it searches for the <nCounter>th occurence of <cStringToMatch> in <cString>. If successful, the sequence will be replaced with <cReplacement>. If <lMode> is set to .F., the same search is performed, but EVERY occurence of <cStringToMatch> till the <nCounter>th (inclusive) will be replaced with <cReplacement>. Note that, in this case, the replacements are performed even if the <nCounter>th occurence does not exist. By using the CSETATMUPA() switch you can decide whether the function restarts searching after a found sequence of after the first character of that sequence. The function allows the use of wildcards in <cStringToMatch> and looks for the settings of SETATLIKE().

Examples

```
? ATREPL("ABC", "ABCDABCDABC", "xx")    --> "xxDxxDxx"
? ATREPL("ABC", "ABCDABC", "ZYXW")      --> "ZYXWDZYXW"
? ATREPL("ABC", "ABCDABCDABC", "xx", 2) --> "xxDxxDABC"
? ATREPL("ABC", "ABCDABCDABC", "xx", 2, .T.)  --> "ABCDxxDABC"
```

Tests

```
ATREPL("ABC", "ABCDABCDABC", "xx") == "xxDxxDxx"
ATREPL("ABC", "ABCDABC", "ZYXW") == "ZYXWDZYXW"
ATREPL("ABC", "ABCDABCDABC", "xx", 2) == "xxDxxDABC"
ATREPL("ABC", "ABCDABCDABC", "xx", 2, .T.) == "ABCDxxDABC"
```

Status

Ready

Compliance

ATREPL() is compatible with CT3's ATREPL(). Note the new, 6th parameter !

Platforms

All

Files

Source is atrepl.c, library is ct3.

See Also:

ARRAY()

CHARLIST()

Generates a list of all characters in a string

Syntax

CHARLIST ([<cString>]) -> cCharacterList

Arguments

[<cString>] is the string for whom the function generates a list of all characters Default: "" (empty string)

Returns

<cCharacterList> a list of the characters in <cString>

Description

The CHARLIST() function generates a list of those characters that are contained in <cString>. This list can contain each character only once, so that its maximum length is 256. The list lists those characters first that are occuring in <cString> first.

Examples

```
? charlist ("Hello World !") --> "Helo Wrd!"
```

Tests

```
charlist ("Hello World !") == "Helo Wrd!"
charlist (nil) == ""
```

Status

Ready

Compliance

CHARLIST() is compatible with CT3's CHARLIST().

Platforms

All

Files

Source is charlist.c, library is libct.

See Also:

CHARNOLIST()
CHARSLIST()
CHARHIST()

CHARSLIST()

Generates a sorted list of all characters in a string

Syntax

CHARSLIST ([<cString>]) -> cSortedCharacterList

Arguments

[<cString>] is the string for whom the function generates a sorted list of all characters Default: "" (empty string)

Returns

<cSortedCharacterList> a sorted list of the characters in <cString>

Description

The CHARLIST() function generates a sorted list of those characters that are contained in <cString>. This list can contain each character only once, so that its maximum length is 256. The function gives the same result as CHARSORT(CHARLIST(<cString>))

Examples

```
? charslist ("Hello World !") --> " !HWdelor"
```

Tests

```
charslist ("Hello World !") == " !HWdelor"
charslist ("Hello World !") == charsort (charlist ("Hello World !"))
charslist (nil) == ""
```

Status

Ready

Compliance

CHARSLIST() is only available in Harbour's CT3 library.

Platforms

All

Files

Source is charlist.c, library is libct.

See Also:

CHARNOLIST()
CHARLIST()
CHARHIST()

CHARNOLIST()

Generates a list of all characters not contained in a string

Syntax

CHARNOLIST ([<cString>]) -> cCharacterList

Arguments

[<cString>] is the string for whom the function generates a list of all characters not contained in that string Default: "" (empty string)

Returns

<cCharacterList> a list of the characters that are not contained in <cString>

Description

The CHARNOLIST() function generates a list of those characters that are not contained in <cString>. This list can contain each character only once, so that its maximum length is 256. The list is alphabetically sorted.

Examples

```
? charnolist (charnolist ("Hello World !")) --> " !HWdelor"
```

Tests

```
charnolist (charnolist ("Hello World !")) == charslist ("Hello World !")
charnolist (charnolist (nil)) == ""
```

Status

Ready

Compliance

CHARNOLIST() is compatible with CT3's CHARNOLIST().

Platforms

All

Files

Source is charlist.c, library is libct.

See Also:

CHARLIST()
CHARSLIST()
CHARHIST()

CHARHIST()

Generates a character histogram of a string

Syntax

```
CHARHIST ([<cString>]) -> aCharacterCount
```

Arguments

[<cString>] is the string for whom the function generates a character histogram Default: "" (empty string)

Returns

<aCharacterCount> an array with 256 elements where the nth element contains the count of character #(n-1) in cString

Description

The CHARHIST() function generates a character histogram of those characters that are contained in <cString>. This histogram is stored in an 256-element array where the nth element contains the count of ASCII character #(n-1) in <cString>.

Examples

```
? charhist ("Hello World !")[109] --> 3  // chr(108)=="l"
```

Tests

```
charhist ("Hello World !")[109] == 3
eval ({||aeval (charhist ("Hello World !"),{|x|nTotal+=x}),nTotal==len("Hello World !")}
```

Status

Ready

Compliance

CHARHIST() is only available in Harbour's CT3 library.

Platforms

All

Files

Source is charlist.c, library is libct.

See Also:

CHARLIST()
CHARNOLIST()
CHARSLIST()

WORDREPL()

Replacement of double characters

Syntax

WORDREPL (<cDoubleCharacterSearchString>, <[@]cString>, <cDoubleCharacterReplaceString>, [<lMode>]) -> cString

Arguments

<cDoubleCharacterSearchString> is a string of double characters that should be replaced <[@]cString> is the processed string <cDoubleCharacterReplaceString> is a string of double characters that replace the one of <cSearchString> [<lMode>] sets the replacement method (see description) Default: .F.

Returns

Description

The WORDREPL() takes the double characters of <cDoubleCharacterSearchString> one after the other and searches for them in <cString>. For <lMode> set to .F., this search is successful, if the double character sequence in <cString> starts at an odd position or at any position, if <lMode> is set to .T. If this happens, the double character sequence will be replaced with the corresponding double character sequence of <cDoubleCharacterReplaceString>. If <cDoubleCharacterReplaceString> is shorter than <cDoubleCharacterSearchString> the last double sequence of <cDoubleCharacterReplaceString> is used for the "rest" of <cDoubleCharacterSearchString>. Note that the last double character sequence in "AABBC" is "BB" in this context !! After the replacement the function restarts the search in <cString> BEHIND the replacement if the CSETATMUPA() switch is turned off, or BEHIND the first character of the replacement if the switch is turned on. (see examples for this !) One can omit the return value of this function by setting the CSETREF() to .T., but one must then pass <cString> by reference to get a result.

Examples

```
? wordrepl("CC", "AABBCCDDEE", "XX") // "AABBXXDDEE"
? wordrepl("aa", "1aaaa", "ba")      // "1abaa"
? wordrepl("aa", "1aaaa", "ba", .T.) // "1baba"
csetatmupa(.T.)
? wordrepl("aa", "1aaaa", "ba")      // "1abaa"
? wordrepl("aa", "1aaaa", "ba", .T.) // "1bbba"
```

Tests

```
wordrepl("CC", "AABBCCDDEE", "XX") == "AABBXXDDEE"
wordrepl("aa", "1aaaa", "ba")      == "1abaa"
wordrepl("aa", "1aaaa", "ba", .T.) == "1baba"
eval ({||csetatmupa(.T.),wordrepl("aa", "1aaaa", "ba")}) == "1abaa"
eval ({||csetatmupa(.T.),wordrepl("aa", "1aaaa", "ba", .T.)}) == "1bbba"
```

Status

Ready

Compliance

WORDREPL() is compatible with CT3's WORDREPL().

Platforms

All

Files

Source is wordrepl.c, library is ct3.

See Also:

CHARREPL()
ARRAY()
POSREPL()
CSETREF()

CSETATMUPA()

ATTOKEN()

Position of a token in a string

Syntax

**ATTOKEN (<cString>, [<cTokenizer>],
[<nTokenCount>], [<nSkipWidth>]) -> nPosition**

Arguments

<cString> is the processed string [<cTokenizer>] is a list of characters separating the tokens in <cString> Default: chr(0)+chr(9)+chr(10)+chr(13)+chr(26)+ chr(32)+chr(32)+chr(138)+chr(141)+ ",.;:!\?/\\<>()#&%+-*" [<nTokenCount>] specifies the count of the token whose position should be calculated Default: last token [<nSkipWidth>] specifies the maximum number of successive tokenizing characters that are combined as ONE token stop, e.g. specifying 1 can yield to empty tokens Default: 0, any number of successive tokenizing characters are combined as ONE token stop

Returns

<nPosition> The start position of the specified token or 0 if such a token does not exist in <cString>.

Description

The ATTOKEN() function calculates the start position of tne <nTokenCount>th token in <cString>. By setting the new <nSkipWidth> parameter to a value different than 0, you can specify how many tokenizing characters are combined at most to one token stop. Be aware that this can result to empty tokens there the start position is not defined clearly. Then, ATTOKEN() returns the position there the token WOULD start if its length is larger than 0. To check for empty tokens, simply look if the character at the returned position is within the tokenizer list.

Examples

```
attoken ("Hello, World!")  --> 8  // empty strings after tokenizer
                                  // are not a token !
```

Tests

```
attoken ("Hello, World!") == 8
attoken ("Hello, World!",,2) == 8
attoken ("Hello, World!",,2,1) == 7
attoken ("Hello, World!"," ",2,1) == 8
```

Status

Ready

Compliance

ATTOKEN() is compatible with CT3's ATTOKEN, but has an additional 4th parameter to let you specify a skip width equal to that in the TOKEN() function.

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

TOKEN()
NUMTOKEN()
TOKENLOWER()
TOKENUPPER()
TOKENSEP()

TOKEN()

Tokens of a string

Syntax

```
TOKEN (<cString>, [<cTokenizer>],
[<nTokenCount>], [<nSkipWidth>],
[<@cPreTokenSep>], [<@cPostTokenSep>]) -> cToken
```

Arguments

<cString> is the processed string [<cTokenizer>] is a list of characters separating the tokens in <cString> Default: chr(0)+chr(9)+chr(10)+chr(13)+chr(26)+ chr(32)+chr(32)+chr(138)+chr(141)+ ",.;:!\?/\\<>()#&%+-*" [<nTokenCount>] specifies the count of the token that should be extracted Default: last token [<nSkipWidth>] specifies the maximum number of successive tokenizing characters that are combined as ONE token stop, e.g. specifying 1 can yield to empty token Default: 0, any number of successive tokenizing characters are combined as ONE token stop [<@cPreTokenSep>] If given by reference, the tokenizer before the actual token will be stored [<@cPostTokenSep>] If given by reference, the tokenizer after the actual token will be stored

Returns

<cToken> the token specified by the parameters given above

Description

The TOKEN() function extracts the <nTokenCount>th token from the string <cString>. In the course of this, the tokens in the string are separated by the character(s) specified in <cTokenizer>. The function may also extract empty tokens, if you specify a skip width other than zero. Be aware of the new 5th and 6th parameter there the TOKEN() function stores the tokenizing character before and after the extracted token. Therefore, additional calls to the TOKENSEP() function are not necessary.

Examples

```
? token ("Hello, World!")         -->  "World"
? token ("Hello, World!",,2,1)    --> ""
? token ("Hello, World!",",",2,1) --> " World!"
? token ("Hello, World!"," ",2,1) --> "World!"
```

Tests

```
token ("Hello, World!") == "World"
token ("Hello, World!",,2,1) == ""
token ("Hello, World!",",",2,1) == " World!"
token ("Hello, World!"," ",2,1) == "World!"
```

Status

Ready

Compliance

TOKEN() is compatible with CT3's TOKEN, but two additional parameters have been added there the TOKEN() function can store the tokenizers before and after the current token.

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

NUMTOKEN()
ATTOKEN()
TOKENLOWER()
TOKENUPPER()

TOKENSEP()

NUMTOKEN()

Retrieves the number of tokens in a string

Syntax

```
NUMTOKEN (<cString>, [<cTokenizer>], [<nSkipWidth>]) -> nTokenCount
```

Tests

```
numtoken ("Hello, World!") ==  2
numtoken ("This is good. See you! How do you do?",".!?") == 3
numtoken ("one,,three,four,,six",",",1) ==  6
```

Status

Ready

Compliance

NUMTOKEN() is compatible with CT3's NUMTOKEN().

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

TOKEN()
ATTOKEN()
TOKENLOWER()
TOKENUPPER()
TOKENSEP()

TOKENLOWER()

Change the first letter of tokens to lower case

Syntax

```
TOKENLOWER (<[@]cString>, [<cTokenizer>], [<nTokenCount>],
[<nSkipWidth>]) -> cString
```

Arguments

<[@]cString> is the processed string [<cTokenizer>] is a list of characters separating the tokens in <cString> Default: chr(0)+chr(9)+chr(10)+chr(13)+chr(26)+ chr(32)+chr(32)+chr(138)+chr(141)+ ",.;:!\?/\\<>()#&%+-*" [<nTokenCount>] specifies the number of tokens that should be processed Default: all tokens [<nSkipWidth>] specifies the maximum number of successive tokenizing characters that are combined as ONE token stop, e.g. specifying 1 can yield to empty token Default: 0, any number of successive tokenizing characters are combined as ONE token stop

Returns

<cString> the string with the lowercased tokens

Description

The TOKENLOWER() function changes the first letter of tokens in <cString> to lower case. To do this, it uses the same tokenizing mechanism as the token() function. If TOKENLOWER() extracts a token that starts with a letter, this letter will be changed to lower case. You can omit the return value of this function by setting the CSETREF() switch to .T., but you must then pass <cString> by reference to get the result.

Examples

```
? tokenlower("Hello, World, here I am!")       // "hello, world, here i am!"
? tokenlower("Hello, World, here I am!",,3)    // "hello, world, here I am!"
? tokenlower("Hello, World, here I am!",",",3) // "hello, World, here I am!"
? tokenlower("Hello, World, here I am!"," W")  // "hello, World, here i am!"
```

Tests

```
tokenlower("Hello, World, here I am!") == "hello, world, here i am!"
tokenlower("Hello, World, here I am!",,3)    == "hello, world, here I am!"
tokenlower("Hello, World, here I am!",",",3) == "hello, World, here I am!"
tokenlower("Hello, World, here I am!"," W")  == "hello, World, here i am!"
```

Status

Ready

Compliance

TOKENLOWER() is compatible with CT3's TOKENLOWER(), but a new 4th parameter, <nSkipWidth> has been added for synchronization with the the other token functions.

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

TOKEN()
NUMTOKEN()
ATTOKEN()
TOKENUPPER()
TOKENSEP()
CSETREF()

TOKENUPPER()

Change the first letter of tokens to upper case

Syntax

TOKENUPPER (<[@]cString>, [<cTokenizer>], [<nTokenCount>], [<nSkipWidth>]) -> cString

Arguments

<[@]cString> is the processed string [<cTokenizer>] is a list of characters separating the tokens in <cString> Default: chr(0)+chr(9)+chr(10)+chr(13)+chr(26)+ chr(32)+chr(32)+chr(138)+chr(141)+ ",.;:!\?/\\<>()#&%+-*" [<nTokenCount>] specifies the number of tokens that should be processed Default: all tokens [<nSkipWidth>] specifies the maximum number of successive tokenizing characters that are combined as ONE token stop, e.g. specifying 1 can yield to empty token Default: 0, any number of successive tokenizing characters are combined as ONE token stop

Returns

<cString> the string with the uppercased tokens

Description

The TOKENUPPER() function changes the first letter of tokens in <cString> to upper case. To do this, it uses the same tokenizing mechanism as the token() function. If TOKENUPPER() extracts a token that starts with a letter, this letter will be changed to upper case. You can omit the return value of this function by setting the CSETREF() switch to .T., but you must then pass <cString> by reference to get the result.

Examples

```
? tokenupper("Hello, world, here I am!")       // "Hello, World, Here I Am!"
? tokenupper("Hello, world, here I am!",,3)    // "Hello, World, Here I am!"
? tokenupper("Hello, world, here I am!",",",3) // "Hello, world, here I am!"
? tokenupper("Hello, world, here I am!"," w")  // "Hello, wOrld, Here I Am!"
```

Tests

```
tokenupper("Hello, world, here I am!")       == "Hello, World, Here I Am!"
tokenupper("Hello, world, here I am!",,3)    == "Hello, World, Here I am!"
tokenupper("Hello, world, here I am!",",",3) == "Hello, world, here I am!"
tokenupper("Hello, world, here I am!"," w")  == "Hello, wOrld, Here I Am!"
```

Status

Ready

Compliance

TOKENUPPER() is compatible with CT3's TOKENUPPER(), but a new 4th parameter, <nSkipWidth> has been added for synchronization with the the other token functions.

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

TOKEN()
NUMTOKEN()
ATTOKEN()
TOKENLOWER()
TOKENSEP()
CSETREF()

TOKENSEP()

Retrieves the token separators of the last token() call

Syntax

TOKENSEP ([<lMode>]) -> cSeparator

Arguments

[<lMode>] if set to .T., the token separator BEHIND the token retrieved from the token() call will be returned. Default: .F., returns the separator BEFORE the token

Returns

retrieved from the last token() call will be returned. These separating characters can now also be retrieved with the token() function.

Description

When one does extract tokens from a string with the token() function, one might be interested in the separator characters that have been used to extract a specific token. To get this information you can either use the TOKENSEP() function after each token() call, or use the new 5th and 6th parameter of the token() function.

Examples

see TOKEN() function

Status

Ready

Compliance

TOKENSEP() is compatible with CT3's TOKENSEP().

Platforms

All

Files

Source is token1.c, library is libct.

See Also:

TOKEN()
NUMTOKEN()
ATTOKEN()
TOKENLOWER()
TOKENUPPER()

CHARMIRR()

Mirror a string

Syntax

CHARMIRR (<[@]cString>, [<lDontMirrorSpaces>]) -> cMirroredString

Arguments

<[@]cString> is the string that should be mirrored
[<lDontMirrorSpaces>] if set to .T., spaces at the end of <cString> will not be mirrored but kept at the end Default: .F., mirror the whole string

Returns

<cMirroredString> the mirrored string

Description

The CHARMIRR() function mirrors a string, i.e. the first character will be put at the end, the second at the last but one position etc.. One can use this function for index searches, but then, the spaces at the end of the string should not be mirrored. One can omit the return value of the function by setting the CSETREF() switch to .T., but <cString> must then be passed by reference to get a result.

Examples

```
? charmirr ("racecar")        // "racecar"
? charmirr ("racecar  ", .T.) // "racecar  "
? charmirr ("racecar  ", .F.) // "  racecar"
```

Tests

```
charmirr ("racecar") == "racecar"
charmirr ("racecar  ", .T.) == "racecar  "
charmirr ("racecar  ", .F.) == "  racecar"
```

Status

Ready

Compliance

CHARMIRR() is compatible with CT3's CHARMIRR().

Platforms

All

Files

Source is charmirr.c, library is ct3.

See Also:

CSETREF()

CHARMIX()

Mix two strings

Syntax

CHARMIX (<cString1>[, <cString2>]) --> cMixedString

Arguments

<cString1> String that will be mixed with the characters from <cString2>
[<cString2>] String whose characters will be mixed with the one from <cString1>. Default: " " (string with one space char)

Returns

<cMixedString> Mixed string

Description

The CHARMIX() function mixes the strings <cString1> and <cString2>. To do this it takes one character after the other alternatively from <cString1> and <cString2> and puts them in the output string. This procedure is stopped when the end of <cString1> is reached. If <cString2> is shorter than <cString1>, the function will start at the begin of <cString2> again. If on the other hand <cString2> is longer than <cString1>, the surplus characters will be omitted.

Examples

```
? CHARMIX("ABC", "123")    // "A1B2C3"
? CHARMIX("ABCDE", "12")   // "A1B2C1D2E1"
? CHARMIX("AB", "12345")   // "A1B2"
? CHARMIX("HELLO", " ")    //  "H E L L O "
? CHARMIX("HELLO", "")     //  "HELLO"
```

Tests

```
CHARMIX("ABC", "123")  == "A1B2C3"
CHARMIX("ABCDE", "12") == "A1B2C1D2E1"
CHARMIX("AB", "12345") == "A1B2"
CHARMIX("HELLO", " ")  == "H E L L O "
CHARMIX("HELLO", "")   == "HELLO"
```

Status

Ready

Compliance

CHARMIX() is compatible with CT3's CHARMIX(). NOTE: CA-Tools version of CHARMIX() will hang if the second parameter is an empty string, this version will not.

Platforms

All

Files

Source is charmix.c, library is ct3.

See Also:

ARRAY()

CHARONE()

Reduce multiple occurences of a character to one

Syntax

CHARONE ([<cCharactersToReduce>,] <cString>) -> cReducedString

Arguments

[<cCharactersToReduce>] specifies the characters the multiple occurences of which should be reduced to one Default: All characters. <cString> specifies the processed string

Returns

<cReducedString>	the string with the reduced occurences

Description

The CHARONE() function reduces multiple occurences of characters in <cString> to a single one. It is important to note that the multiple occurences must occur directly one behind the other. This behaviour is is in contrast to the CHARLIST() function.

Examples

```
? CHARONE("122333a123")      // "123a123"
? CHARONE("A  B  CCCD")      // "A B CD"
? CHARONE(" ", "A  B  A  B") // "A B A B"
? CHARONE("o", "122oooB12o") // "122oB12o"
```

Tests

```
CHARONE("122333a123")      == "123a123"
CHARONE("A  B  CCCD")      == "A B CD"
CHARONE(" ", "A  B  A  B") == "A B A B"
CHARONE("o", "122oooB12o") == "122oB12o"
```

Status

Ready

Compliance

CHARONE() is compatible with CT3's CHARONE().

Platforms

All

Files

Source is charone.c, library is ct3.

See Also:

ARRAY()

WORDONE()

Reduce multiple occurences of a double character to one

Syntax

WORDONE ([<cDoubleCharactersToReduce>,] <cString>) -> cReducedString

Arguments

[<cDoubleCharactersToReduce>] specifies the double characters the multiple occurences of which should be reduced to one Default: All characters. <cString> specifies the processed string

Returns

<cReducedString> the string with the reduced occurences

Description

The WORDONE() function reduces multiple occurences of double characters in <cString> to a single one. It is important to note that the multiple occurences must occur directly one behind the other.

Examples

```
? WORDONE("12ABAB12")       // "12AB12"
? WORDONE("1AAAA2")         // "1AAAA2"
? WORDONE("12", "1212ABAB") // "12ABAB"
```

Tests

```
WORDONE("12ABAB12")       == "12AB12"
WORDONE("1AAAA2")         == "1AAAA2"
WORDONE("12", "1212ABAB") == "12ABAB"
```

Status

Ready

Compliance

WORDONE() is compatible with CT3's WORDONE().

Platforms

All

Files

Source is charone.c, library is ct3.

See Also:

ARRAY()

CHARONLY()

Intersectional set of two strings based on characters

Syntax

CHARONLY (<cThisCharactersOnly>, <cString>) -> cReducedString

Arguments

<cThisCharactersOnly>   specifies the characters that must not be deleted in <cString>. <cString>   is the string that should be processed

Returns

<cReducedString>   A string with all characters deleted but those specified in <cThisCharactersOnly>.

Description

The CHARONLY() function calculates the intersectional set of two strings. To do this, it deletes all characters from <cString> that do not appear in <cThisCharacterOnly>.

Examples

```
? CHARONLY("0123456789", "0211 - 38 99 77")  //  "0211389977"
? CHARONLY("0123456789", "0211/ 389 977")    //  "0211389977"
```

Tests

```
CHARONLY("0123456789", "0211 - 38 99 77") == "0211389977"
CHARONLY("0123456789", "0211/ 389 977")   == "0211389977"
```

Status

Ready

Compliance

CHARONLY() is compatible with CT3's CHARONLY().

Platforms

All

Files

Source is charonly.c, library is ct3.

See Also:

ARRAY()

WORDONLY()

Intersectional set of two strings based on double characters

Syntax

WORDONLY (<cThisDoubleCharactersOnly>, <cString>) -> cReducedString

Arguments

<cThisDoubleCharactersOnly> specifies the double characters that must not be deleted in <cString>. <cString> is the string that should be processed

Returns

<cReducedString> A string with all double characters deleted but those specified in <cThisCharactersOnly>.

Description

The WORDONLY() function calculates the intersectional set of two strings based on double characters. To do this, it deletes all double characters from <cString> that do not appear in <cThisDoubleCharacterOnly>.

Examples

```
? WORDONLY("AABBCCDD", "XXAAYYBBZZ")  // "AABB"
? WORDONLY("AABBCCDD", "XAAYYYBBZZ")  // "BB"
```

Tests

```
WORDONLY("AABBCCDD", "XXAAYYBBZZ") == "AABB"
WORDONLY("AABBCCDD", "XAAYYYBBZZ") == "BB"
```

Status

Ready

Compliance

WORDONLY() is compatible with CT3's WORDONLY().

Platforms

All

Files

Source is charonly.c, library is ct3.

See Also:

ARRAY()

CHARREM()

Removes characters from a string

Syntax

```
CHARREM (<cDeleteThisCharacters>, <cString>) -> cReducedString
```

Arguments

<cDeleteThisCharacters> specifies the characters that should be deleted in <cString> <cString>) is the string that should be processed

Returns

<cReducedString> is a string where the characters specified in <cDeleteThisCharacters> are deleted

Description

The CHARREM() function deletes the characters specified in <cDeleteThisCharacters> from <cString>.

Examples

```
? CHARREM(" ", " 1  2  ")   // "12"
? CHARREM("3y", "xyz123")   // "xz12"
```

Tests

```
CHARREM(" ", " 1  2  ") == "12"
CHARREM("3y", "xyz123") == "xz12"
```

Status

Ready

Compliance

CHARREM() is compatible with CT3's CHARREM().

Platforms

All

Files

Source is charonly.c, library is ct3.

See Also:

ARRAY()

WORDREM()

Removes characters from a string

Syntax

WORDREM (<cDeleteThisDoubleCharacters>, <cString>) -> cReducedString

Arguments

<cDeleteThisDoubleCharacters> specifies the double characters that should be deleted in <cString> <cString>) is the string that should be processed

Returns

<cReducedString> is a string where the double characters specified in <cDeleteThisDoubleCharacters> are deleted

Description

The WORDREM() function deletes the double characters specified in <cDeleteThisDoubleCharacters> from <cString>.

Examples

```
? WORDREM("abcd", "0ab1cd")   // "0ab1"
? WORDREM("abcd", "ab0cd1")   // "0cd1"
```

Tests

```
WORDREM("abcd", "0ab1cd") == "0ab1"
WORDREM("abcd", "ab0cd1") == "0cd1"
```

Status

Ready

Compliance

WORDREM() is a new function available only in Harbour's CT3.

Platforms

All

Files

Source is charonly.c, library is ct3.

See Also:

ARRAY()

CHARREPL()

Replacement of characters

Syntax

CHARREPL (<cSearchString>, <[@]cString>, <cReplaceString>, [<lMode>]) -> cString

Arguments

<cSearchString> is a string of characters that should be replaced <[@]cString> is the processed string <cReplaceString> is a string of characters that replace the one of <cSearchString> [<lMode>] sets the replacement method (see description) Default: .F.

Returns

<cString> the processed string

Description

The CHARREPL() function replaces certain characters in <cString> with others depending on the setting of <lMode>. If <lMode> is set to .F., the function takes the characters of <cSearchString> one after the other, searches for them in <cString> and, if successful, replaces them with the corresponding character of <cReplaceString>. Be aware that if the same characters occur in both <cSearchString> and <cReplaceString>, the character on a certain position in <cString> can be replaced multiple times. if <lMode> is set to .T., the function takes the characters in <cString> one after the other, searches for them in <cSearchString> and, if successful, replaces them with the corresponding character of <cReplaceString>. Note that no multiple replacements are possible in this mode. If <cReplaceString> is shorter than <cSearchString>, the last character of <cReplaceString> is used as corresponding character for the the "rest" of <cSearchString>. One can omit the return value by setting the CSETREF() switch to .T., but then one must pass <cString> by reference to get the result.

Examples

```
? charrepl ("1234", "1x2y3z", "abcd")             // "axbycz"
? charrepl ("abcdefghij", "jhfdb", "1234567890") // "08642"
? charrepl ("abcdefghij", "jhfdb", "12345")      // "55542"
? charrepl ("1234", "1234", "234A")              // "AAAA"
? charrepl ("1234", "1234", "234A", .T.)         // "234A"
```

Tests

```
charrepl ("1234", "1x2y3z", "abcd") == "axbycz"
charrepl ("abcdefghij", "jhfdb", "1234567890") == "08642"
charrepl ("abcdefghij", "jhfdb", "12345") == "55542"
charrepl ("1234", "1234", "234A") == "AAAA"
charrepl ("1234", "1234", "234A", .T.) == "234A"
```

Status

Ready

Compliance

CHARREPL() is compatible with CT3's CHARREPL().

Platforms

All

Files

Source is charrepl.c, library is ct3.

See Also:

CSETREF()

CHARSORT()

Sort sequences within a string.

Syntax

CHARSORT (<[@]cString>, [<nElementLength>], [<nCompareLength>], [<nIgnoreCharacters>], [<nElemenOffset>], [<nSortLength>], [<lDescending>]) -> cSortedString

Arguments

<[@]cString> is the string that should be processed [<nElementLength>] specifies the length of the elements that should be sorted Default: 1 [<nCompareLength>] specifies how many characters within one element should be used for comparison Default: <nElementLength> [<nIgnoreCharacters>] specifies the number of characters at the beginning of <cString> that should be ignored in the sort process Default: 0 [<nElementOffset>] specifies the offset of the comparison string within a element Default: 0 [<nSortLength>] specifies how many characters in <cString>, starting from the <nIgnoreCharacters> position, should be sorted Default: len(cString)-nIgnoreCharacters [<lDescending>]) specifies whether the process should sort descending or not

Returns

<cSortedString> the string resulting from the sort process

Description

The CHARSORT function sorts the characters within a string <cString>. With the parameters <nIgnoreCharacters> and <nSortLength>, you can determine that only the substring from position <nIgnoreCharacters>+1 to position <nIgnoreCharacters>+<nSortLength> within <cString> should be sorted. The sorting algorithm is determined with the other parameters. <nElementLength> specifies the length of one element, i.e. there are <nSortLength>/<nElementLength> elements that are sorted. Note that surplus characters are not sorted but stay at their position. To do the sorting, the function uses the Quicksort algorithm implemented in the C-lib qsort() function. This algorithm needs to know how to compare and order two elements. This is done by comparing the ASCII values of a substring within each element. This substring is determined by the parameters <nElementOffset> and <nCompareLength> and the order by <lDescending>. By setting the CSETREF() switch to .T., one can omit the return value of the function, but one must then pass <cString> by reference.

Examples

```
? CHARSORT("qwert")                     // "eqrtw"
? CHARSORT("qwert", 2)                  // "erqwt"
? CHARSORT("b1a4a3a2a1", 2, 1)          // "a2a1a3a4b1"
? CHARSORT("XXXqwert", 1, 1, 3)         // "XXXeqrtw"
? CHARSORT("b1a4a3a2a1", 2, 1, 0, 1)    // "a1b1a2a3a4"
? CHARSORT("384172852", 1, 1, 0, 0, 4)  // "134872852"
? CHARSORT("qwert", .T.)                // "wtrqe"
```

Tests

```
CHARSORT("qwert")                     == "eqrtw"
CHARSORT("qwert", 2)                  == "erqwt"
CHARSORT("b1a4a3a2a1", 2, 1)          == "a2a1a3a4b1"
CHARSORT("XXXqwert", 1, 1, 3)         == "XXXeqrtw"
CHARSORT("b1a4a3a2a1", 2, 1, 0, 1)    == "a1b1a2a3a4"
CHARSORT("384172852", 1, 1, 0, 0, 4)  == "134872852"
CHARSORT("qwert", .T.)                == "wtrqe"
```

Status

Ready

Compliance

CHARSORT() is compatible with CT3's CHARSORT().

Platforms

All

Files

Source is charsort.c, library is ct3.

See Also:

CSETREF()

CHARSWAP()

Swap neighbouring characters in a string

Syntax

CHARSWAP (<[@]cString>) -> cSwappedString

Arguments

<[@]cString>	is the string that should be processed

Returns

<cSwappedString>	a string where neighbour characters are swapped

Description

The CHARSWAP() function loops through <cString> in steps of two characters and exchanges the characters from the odd and the even positions. By setting the CSETREF() switch to .T., one can omit the return value of this functin, but one must then pass <cString> by reference.

Examples

```
? CHARSWAP("0123456789")   // "1032547698"
? CHARSWAP("ABCDEFGHIJK")  // "BADCFEHGJIK"
```

Tests

```
CHARSWAP("0123456789")  == "1032547698"
CHARSWAP("ABCDEFGHIJK") == "BADCFEHGJIK"
```

Status

Ready

Compliance

CHARSWAP() is compatible with CT3's CHARSWAP().

Platforms

All

Files

Source is charswap.c, library is libct.

See Also:

WORDSWAP()
CSETREF()

WORDSWAP()

Swap neighbouring double characters in a string

Syntax

WORDSWAP (<[@]cString> [, <lSwapCharacters>]) -> cSwappedString

Arguments

<[@]cString> is the string that should be processed
[<lSwapCharacters>] specifies whether an additional swap should be done within the double characters Default: .F., no additional swap

Returns

<cSwappedString> a string where neighbouring double characters are swapped

Description

The WORDSWAP() function loops through <cString> in steps of four characters and exchanges the double characters from the first and second position with the one from the third and forth position. Additionally the function can perform a swap of the both char of each double character. By setting the CSETREF() switch to .T., one can omit the return value of this functin, but one must then pass <cString> by reference.

Examples

```
? WORDSWAP("1234567890")        // "3412785690"
? WORDSWAP("1234567890", .t.)   // "4321876590"
```

Tests

```
WORDSWAP("1234567890")      == "3412785690"
WORDSWAP("1234567890", .t.) == "4321876590"
```

Status

Ready

Compliance

WORDSWAP() is compatible with CT3's WORDSWAP().

Platforms

All

Files

Source is charswap.c, library is libct.

See Also:

CHARSWAP()
CSETREF()

JUSTLEFT()

Move characters from the beginning to the end of a string

Syntax

JUSTLEFT (<[@]cString>, [<cChar>|<nChar>]) -> cJustifiedString

Description

TODO: add documentation

Status

Started

Compliance

JUSTLEFT() is compatible with CT3's JUSTLEFT().

Platforms

All

Files

Source is justify.c, library is libct.

See Also:

JUSTRIGHT()

JUSTRIGHT()

Move characters from the end to the beginning of a string

Syntax

```
JUSTRIGHT (<[@]cString>, [<cChar>|<nChar>]) -> cJustifiedString
```

Description

TODO: add documentation

Status

Started

Compliance

JUSTRIGHT() is compatible with CT3's JUSTRIGHT().

Platforms

All

Files

Source is justify.c, library is libct.

See Also:

JUSTLEFT()

TOKENINIT()

Initializes a token environment

Syntax

TOKENINIT (<[@]cString>], [<cTokenizer>], [<nSkipWidth>], [<@cTokenEnvironment>]) -> lState

Arguments

<[@]cString> is the processed string <cTokenizer> is a list of characters separating the tokens in <cString> Default: chr(0)+chr(9)+chr(10)+chr(13)+chr(26)+ chr(32)+chr(32)+chr(138)+chr(141)+ ",.;:!\?/\\<>()#&%+-*" <nSkipWidth> specifies the maximum number of successive tokenizing characters that are combined as ONE token stop, e.g. specifying 1 can yield to empty token Default: 0, any number of successive tokenizing characters are combined as ONE token stop <@cTokenEnvironment> is a token environment stored in a binary encoded string

Returns

<lState> success of the initialization

Description

The TOKENINIT() function initializes a token environment. A token environment is the information about how a string is to be tokenized. This information is created in the process of tokenization of the string <cString> - equal to the one used in the TOKEN() function with the help of the <cTokenizer> and <nSkipWidth> parameters.

This token environment can be very useful when large strings have to be tokenized since the tokenization has to take place only once whereas the TOKEN() function must always start the tokenizing process from scratch.

Unlike CTIII, this function provides two mechanisms of storing the resulting token environment. If a variable is passed by reference as 4th parameter, the token environment is stored in this variable, otherwise the global token environment is used. Do not modify the token environment string directly !

Additionally, a counter is stored in the token environment, so that the tokens can successivly be obtained. This counter is first set to 1. When the TOKENINIT() function is called without a string a tokenize, the counter of either the global environment or the environment given by reference in the 4th parameter is rewind to 1.

Additionally, unlike CTIII, tokeninit() does not need the string <cString> to be passed by reference, since one must provide the string in calls to TOKENNEXT() again.

Examples

```
tokeninit (cString)   // tokenize the string <cString> with default
                      // rules and store the token environment globally
                      // and eventually delete an old global TE
tokeninit (@cString)  // no difference in result, but eventually faster,
                      // since the string must not be copied
tokeninit()           // rewind counter of global TE to 1
tokeninit ("1,2,3",",",1) // tokenize constant string, store in global TE
tokeninit (cString,,1,@cTE1)  // tokenize cString and store TE in
                              // cTE1 only without overriding global TE
tokeninit (cString,,1,cTE1)  // tokenize cString and store TE in
                             // GLOBAL TE since 4th parameter is
                             // not given by reference !!!
tokeninit (,,,@cTE1)         // set counter in TE stored in cTE1 to 1
```

Status

Ready

Compliance

TOKENINIT() is compatible with CTIII's TOKENINIT(), but there is an additional parameter featuring local token environments.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKEN()
TOKENEXIT()
TOKENNEXT()
TOKENNUM()
TOKENAT()
SAVETOKEN()
RESTTOKEN()
TOKENEND()

TOKENNEXT()

Successivly obtains tokens from a string

Syntax

TOKENNEXT (<[@]cString>, [<nToken>], [<@cTokenEnvironment>]) -> cToken

Arguments

<[@]cString> the processed string <nToken> a token number <@cTokenEnvironment> a token environment

Returns

<cToken> a token from <cString>

Description

With TOKENNEXT(), the tokens determined with the TOKENINIT() functions can be retrieved. To do this, TOKENNEXT() uses the information stored in either the global token environment or the local one supplied by <cTokenEnvironment>. Note that, is supplied, this 3rd parameter has always to be passed by reference.

If the 2nd parameter, <nToken> is given, TOKENNEXT() simply returns the <nToken>th token without manipulating the TE counter. Otherwise the token pointed to by the TE counter is returned and the counter is incremented by one. Like this, a simple loop with TOKENEND() can be used to retrieve all tokens of a string successivly.

Note that <cString> does not have to be the same used in TOKENINIT(), so that one can do a "correlational tokenization", i.e. tokenize a string as if it was another! E.G. using TOKENINIT() with the string "AA,BBB" but calling TOKENNEXT() with "CCCEE" would give first "CC" and then "EE" (because "CCCEE" is not long enough).

Examples

```
// default behavhiour
tokeninit (cString) // initialize a TE
do while (!tokenend())
  ? tokennext (cString)  // get all tokens successivly
enddo
? tokennext (cString, 3)  // get the 3rd token, counter will remain the same
tokenexit()              // free the memory used for the global TE
```

Status

Ready

Compliance

TOKENNEXT() is compatible with CTIII's TOKENNEXT(), but there are two additional parameters featuring local token environments and optional access to tokens.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNUM()
TOKENAT()
SAVETOKEN()
RESTTOKEN()
TOKENEND()

TOKENNUM()

Get the total number of tokens in a token environment

Syntax

TOKENNUM ([<@cTokenEnvironment>]) -> nNumberofTokens

Arguments

<@cTokenEnvironment>	a token environment

Returns

<nNumberofTokens>	number of tokens in the token environment

Description

The TOKENNUM() function can be used to retrieve the total number of tokens in a token environment. If the parameter <@cTokenEnvironment> is supplied (must be by reference), the information from this token environment is used, otherwise the global TE is used.

Examples

```
tokeninit ("a.b.c.d", ".", 1)  // initialize global TE
? tokennum()  // --> 4
```

Status

Ready

Compliance

TOKENNUM() is a new function in Harbour's CTIII library.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNEXT()
TOKENAT()
SAVETOKEN()
RESTTOKEN()
TOKENEND()

TOKENEND()

Check whether additional tokens are available with TOKENNEXT()

Syntax

TOKENEND ([<@cTokenEnvironment>]) -> lTokenEnd

Arguments

<@cTokenEnvironment>	a token environment

Returns

<lTokenEnd>	.T., if additional tokens are available

Description

The TOKENEND() function can be used to check whether the next call to TOKENNEXT() would return a new token. This can not be decided with TOKENNEXT() alone, since an empty token cannot be distinguished from a "no more" tokens. If the parameter <@cTokenEnvironment> is supplied (must be by reference), the information from this token environment is used, otherwise the global TE is used. With a combination of TOKENEND() and TOKENNEXT(), all tokens from a string can be retrieved successivly (see example).

Examples

```
tokeninit ("a.b.c.d", ".", 1)  // initialize global TE
do while (!tokenend())
  ? tokennext ("a.b.c.d")  // get all tokens successivly
enddo
```

Status

Ready

Compliance

TOKENEND() is compatible with CTIII's TOKENEND(), but there are is an additional parameter featuring local token environments.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNEXT()
TOKENNUM()
TOKENAT()
SAVETOKEN()
RESTTOKEN()

TOKENEXIT()

Release global token environment

Syntax

TOKENEXIT () -> lStaticEnvironmentReleased

Returns

<lStaticEnvironmentReleased> .T., if global token environment is successfully released

Description

The TOKENEXIT() function releases the memory associated with the global token environment. One should use it for every tokeninit() using the global TE. Additionally, TOKENEXIT() is implicitly called from CTEXIT() to free the memory at library shutdown.

Examples

```
tokeninit (cString) // initialize a TE
do while (!tokenend())
  ? tokennext (cString)  // get all tokens successivly
enddo
? tokennext (cString, 3)  // get the 3rd token, counter will remain the same
tokenexit()              // free the memory used for the global TE
```

Status

Ready

Compliance

TOKENEXIT() is a new function in Harbour's CTIII library.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

- TOKENINIT()
- TOKENNEXT()
- TOKENNUM()
- TOKENAT()
- SAVETOKEN()
- RESTTOKEN()
- TOKENEND()

TOKENAT()

Get start and end positions of tokens in a token environment

Syntax

TOKENAT ([<lSeparatorPositionBehindToken>], [<nToken>],
[<@cTokenEnvironment>]) -> nPosition

Arguments

<lSeparatorPositionBehindToken> .T., if TOKENAT() should return the position of the separator character BEHIND the token. Default: .F., return start position of a token. <nToken> a token number <@cTokenEnvironment> a token environment

Returns

Description

The TOKENAT() function is used to retrieve the start and end position of the tokens in a token environment. Note however that the position of last character of a token is given by tokenat (.T.)-1 !!

If the 2nd parameter, <nToken> is given, TOKENAT() returns the positions of the <nToken>th token. Otherwise the token pointed to by the TE counter, i.e. the token that will be retrieved by TOKENNEXT() _NEXT_ is used.

If the parameter <@cTokenEnvironment> is supplied (must be by reference), the information from this token environment is used, otherwise the global TE is used.

Tests

```
tokeninit (cString) // initialize a TE
do while (!tokenend())
  ? "From", tokenat(), "to", tokenat(.T.)-1
  ? tokennext (cString)  // get all tokens successivly
enddo
? tokennext (cString, 3)  // get the 3rd token, counter will remain the same
tokenexit()               // free the memory used for the global TE
```

Status

Ready

Compliance

TOKENAT() is compatible with CTIII's TOKENAT(), but there are two additional parameters featuring local token environments and optional access to tokens.

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNEXT()
TOKENNUM()
SAVETOKEN()
RESTTOKEN()
TOKENEND()

SAVETOKEN()

Save the global token environment

Syntax

SAVETOKEN () -> cStaticTokenEnvironment

Returns

<cStaticTokenEnvironment>   a binary string encoding the global TE

Description

The SAVETOKEN() function can be used to store the global TE for future use or when two or more incremental tokenizers must the nested. Note however that the latter can now be solved with locally stored token environments.

Status

Ready

Compliance

SAVETOKEN() is compatible with CTIII's SAVETOKEN(),

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNEXT()
TOKENNUM()
TOKENAT()
RESTTOKEN()
TOKENEND()

RESTTOKEN()

Restore global token environment

Syntax

RESTTOKEN (<cStaticTokenEnvironment>) -> cOldStaticEnvironment

Arguments

<cStaticTokenEnvironment>	a binary string encoding a TE

Returns

<cOldStaticEnvironment>	a string encoding the old global TE

Description

The RESTTOKEN() function restores the global TE to the one encoded in <cStaticTokenEnvironment>. This can either be the return value of SAVETOKEN() or the value stored in the 4th parameter in a TOKENINIT() call.

Status

Ready

Compliance

RESTTOKEN() is compatible with CTIII's RESTTOKEN(),

Platforms

All

Files

Source is token2.c, library is libct.

See Also:

TOKENINIT()
TOKENEXIT()
TOKENNEXT()
TOKENNUM()
TOKENAT()
SAVETOKEN()
TOKENEND()

SETMATHERR()

Sets the math error correction status and mode

Syntax

SETMATHERR ([<nStatus>] [,<[@]nMode>]) -> nOldStatus

Arguments

[<nStatus>] new math error correction status [<[@]nMode>] new math error correction mode OR placeholder for current mode (if passed by reference)

Returns

math error correction

Description

Most math functions within the CT3 library (and in Harbour itself) rely on the standard C math library which, on some platforms, calls a certain, user-definable error handling routine when one of the following mathematical errors occur (constants defined in cterror.ch):

CT_ERROR_MATHLIB unknown math lib error CT_ERROR_MATHLIB_DOMAIN a domain error has occured, such as sqrt (-1) CT_ERROR_MATHLIB_SING a singularity will result, such as pow (0, -2) CT_ERROR_MATHLIB_OVERFLOW an overflow will result, such as pow (10, 100) CT_ERROR_MATHLIB_UNDERFLOW an underflow will result, such as pow (10, -100) CT_ERROR_MATHLIB_TLOSS total loss of significance will result, such as exp (1000) CT_ERROR_MATHLIB_PLOSS partial loss of significance will result, such as sin (10e70)

The CT3 library redirects these errors within its math routines to its own math handler. The behaviour of this handler depends on the values of <nStatus> and <nMode>:

The values of <nStatus> and <nOldStatus> specify whether the CT3 math handler is active. It can be one of the following values (defined in ct.ch):

CT_MATHERR_STATUS_NOTFOUND	math handler is not installed
CT_MATHERR_STATUS_INACTIVE	math handler is installed but inactive
CT_MATHERR_STATUS_ACTIVE	math handler is installed and active

Be aware that, if CT_MATHERR_STATUS_NOTFOUND is used as argument, SETMATHERR() will NOT deinstall the math handler. The math handler is installed by CTINIT(), remains inactive at first, and is deinstalled by CTEXIT().

The value of <nMode> specifies the behaviour of the CT3 math handler if it is installed and active. It can be one of the following values:

CT_MATHERR_MODE_NONE no correction at all, program will exit
CT_MATHERR_MODE_DEFAULT default return value will be used, no error msgs !
CT_MATHERR_MODE_USER error will be thrown to user who is responsible for error correction CT_MATHERR_MODE_USERDEFAULT error will be thrown, but if user fails, default correction will be used

The default behaviour is CT_MATHERR_MODE_DEFAULT.

Be aware that, if <nMode> is passed by reference, SETMATHERR() will store the current value in <@nMode> rather than setting a new one.

Status

Ready
SETMATHERR() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ctmath.c, library is ct3.

See Also:

ARRAY( )

SETPREC()

Set precision of math functions

Syntax

SETPREC (<nPrecision>) -> cEmptyString

Arguments

<nPrecision>	digit count between 1 and 16, defaults to 16

Returns

Description

Be aware that calls to this functions do _NOT_ affect the calculation precision of the math functions at the moment.

Status

Ready

Compliance

SETPREC() is compatible with CT3's SETPREC.

Platforms

All

Files

Source is ctmath.c, library is ct3.

GETPRECO()

Get precision of math functions

Syntax

```
GETPREC () -> nDigits
```

Returns

Description

Be aware that calls to this functions do _NOT_ affect the calculation precision of the math functions at the moment.

Status

Ready

Compliance

GETPREC() is compatible with CT3's GETPREC.

Platforms

All

Files

Source is ctmath.c, library is ct3.

FV()
Future value of a capital

Syntax

FV (nDeposit, nInterest, nPeriods) --> nFutureValue

Arguments

<nDeposit> amount of money invested per period <nInterest> rate of interest per period, 1 == 100% <nPeriods> period count

Returns

<nFutureValue> Total value of the capital after <nPeriods> of paying <nDeposit> and <nInterest> interest being paid every period and added to the capital (resulting in compound interest)

Description

FV() calculates the value of a capital after <nPeriods> periods. Starting with a value of 0, every period, <nDeposit> (Dollars, Euros, Yens, ...) and an interest of <nInterest> for the current capital are added for the capital (<nInterest>=Percent/100). Thus, one gets the non-linear effects of compound interests: value in period 0 = 0 value in period 1 = ((value in period 0)*(1+<nInterest>/100)) + <nDeposit> value in period 2 = ((value in period 1)*(1+<nInterest>/100)) + <nDeposit> etc.... value in period <nPeriod> = ((value in period <nPeriod>-1)*(1+<nInterest>/100))< + <nDeposit> = <nDeposit> * sum from i=0 to <nPeriod>-1 over (1+<nInterest>/100)^i = <nDeposit> * ((1+<nInterest>/100)^n-1) / (<nInterest>/100)

Examples

```
// Payment of 1000 per year for 10 years at a interest rate
// of 5 per cent per year

? fv (1000, 0.05, 10)  --> 12577.893
```

Tests

```
fv (1000, 0.00, 10) == 10000.0
fv (1000, 0.05, 10) == 12577.893
```

Status

Ready

Compliance

FV() is compatible with CT3's FV().

Platforms

All

Files

Source is finan.c, library is libct.

See Also:

PV()
PAYMENT()
PERIODS()
RATE()

PV()

Present value of a loan

Syntax

PV (nPayment, nInterest, nPeriods) --> nPresentValue

Arguments

<nPayment> amount of money paid back per period <nInterest> rate of interest per period, 1 == 100% <nPeriods> period count

Returns

<nPresentValue> Present value of a loan when one is paying back <nDeposit> per period at a rate of interest of <nInterest> per period

Description

PV() calculates the present value of a loan that is paid back in <nPeriods> payments of <nPayment> (Dollars, Euros, Yens,...) while the rate of interest is <nInterest> per period: debt in period 0 = <nPresentValue> debt in period 1 = ((debt in period 0)-<nPayment>)*(1+<nInterest>/100) debt in period 2 = ((debt in period 1)-<nPayment>)*(1+<nInterest>/100) etc... debt in period <nPeriod> = ((debt in period <nPeriod>-1)-<nPayment>)*(1+<nInterest>/100) -> has to be 0, so <nPresentValue> = <nPayment>*(1-(1+<nInterest>/100)^(-n))/(<nInterest>/100)

Examples

```
// You can afford to pay back 100 Dollars per month for 5 years
// at a interest rate of 0.5% per month (6% per year), so instead
// of 6000 Dollars (the amount you will pay back) the bank will pay
// you

? pv (100, 0.005, 60)  --> 5172.56
```

Tests

```
pv (100, 0.0, 60)   == 6000.0
pv (100, 0.005, 60) == 5172.56
```

Status

Ready

Compliance

PV() is compatible with CT3's PV().

Platforms

All

Files

Source is finan.c, library is libct.

See Also:

FV()
PAYMENT()
PERIODS()
RATE()

PAYMENT()

Payments for a loan

Syntax

PAYMENT (nLoan, nInterest, nPeriods) --> nPayment

Arguments

<nLoan> amount of money you get from the bank <nInterest> rate of interest per period, 1 == 100% <nPeriods> period count

Returns

<nPayment> Periodical payment one has to make to pay the loan <nLoan> back

Description

PAYMENT() calculates the payment one has to make periodically to pay back a loan <nLoan> within <nPeriods> periods and for a rate of interest <nInterest> per period. debt in period 0 = <nLoan> debt in period 1 = ((debt in period 0)-<nPayment>)*(1+<nInterest>/100) debt in period 2 = ((debt in period 1)-<nPayment>)*(1+<nInterest>/100) etc... debt in period <nPeriod> = ((debt in period <nPeriod>-1)-<nPayment>)*(1+<nInterest>/100) -> has to be 0, so <nPayment> = <nLoan>*(<nInterest>/100)/(1-(1+<nInterest>/100)^(-n))

Examples

```
// You get a loan of 5172.56 at a interest rate of 0.5% per
// month (6% per year).
// For 5 years, you have to pay back every month

? payment (5172.56, 0.005, 60)  --> 100.00
```

Tests

```
payment (5172.56, 0.0, 60)   == 86.21
payment (5172.56, 0.005, 60) == 100.00
```

Status

Ready

Compliance

PAYMENT() is compatible with CT3's PAYMENT().

Platforms

All

Files

Source is finan.c, library is libct.

See Also:

PV()
FV()
PERIODS()
RATE()

PERIODS()

Number of periods for a loan

Syntax

PERIODS (nLoan, nPayment, nInterest) --> nPeriods

Arguments

<nLoan> amount of money you get from the bank <nPayment> amount of money you pay back per period <nInterest> rate of interest per period, 1 == 100%

Returns

<nPeriods> number of periods you need to pay the loan back

Description

PERIODS() calculates the number of periods one needs to pay back a loan of <nLoan> with periodical payments of <nPayment> and for a rate of interest <nInterest> per period. debt in period 0 = <nLoan> debt in period 1 = ((debt in period 0)-<nPayment>)*(1+<nInterest>/100) debt in period 2 = ((debt in period 1)-<nPayment>)*(1+<nInterest>/100) etc... debt in period <nPeriod> = ((debt in period <nPeriod>-1)-<nPayment>)*(1+<nInterest>/100) -> has to be 0, so <nPeriods> = -log(1-<nLoan>*(<nInterest>/100)/<nPayment>)/log(1+<nInterest>/100))

Note, however that in the case of nPayment <= <nLoan>*(<nInterest>/100), one would need infinite time to pay the loan back. The functions does then return -1.

Examples

```
// You get a loan of 5172.56 at a interest rate of 0.5% per
// month (6% per year).
// You can afford to pay 100 back every month, so you need

? periods (5172.56, 100, 0.005)  --> 60.0

// months to cancel the loan.
```

Tests

```
periods (5172.56, 100, 0.005) == 60.0
periods (5172.56, 100, 0.0) == 51.7256
```

Status

Ready

Compliance

PERIODS() is compatible with CT3's PERIODS().

Platforms

All

Files

Source is finan.c, library is libct.

See Also:

PV()
FV()
PAYMENT()
RATE()

RATE()

Estimate rate of interest for a loan

Syntax

```
RATE (nLoan, nPayment, nPeriods) --> nRate
```

Arguments

<nLoan>   amount of money you get from the bank <nPayment> amount of money you pay back per period <nPeriods>   number of periods you pay the loan back

Returns

<nInterest>   estimated rate of interest per period, 1 == 100%

Description

RATE() calculates the rate of interest per period for the given loan, payment per periods and number of periods. This is done with the same equation used in the PAYMENT() or PERIODS() function:

```
<nPayment> = <nLoan>*(<nInterest>/100)/(1-(1+<nInterest>/100)^(-<nPeriods>))
```

However, this equation can not be solved for <nInterest> in a "closed" manner, i.e. <nInterest> = ..., so that the result can only be estimated.

Examples

```
// You get a loan of 5172.56, pay 100 back every month for
// 5 years (60 months). The effective interest rate per
// period (=month) is

? rate (5172.56, 100, 60)  --> 0.005
```

Tests

```
rate (5172.56, 100, 60.0) == 0.005
rate (6000.0, 100, 60.0) == 0.0
```

Status

Ready

Compliance

RATE() is compatible with CT3's RATE().

Platforms

All

Files

Source is finan.c, library is libct.

See Also:

PV()
FV()
PAYMENT()
PERIODS()

COUNTLEFT()

Count a certain character at the beginning of a string

Syntax

COUNTLEFT (<cString>, [<cSearch|nSearch>]) -> nCount

Description

TODO: add documentation

Status

Started

Compliance

COUNTLEFT() is compatible with CT3's COUNTLEFT().

Platforms

All

Files

Source is count.c, library is libct.

See Also:

COUNTRIGHT()

COUNTRIGHT()

Count a certain character at the end of a string

Syntax

COUNTRIGHT (<cString>, [<cSearch|nSearch>]) -> nCount

Description

TODO: add documentation

Status

Started

Compliance

COUNTRIGHT() is compatible with CT3's COUNTRIGHT().

Platforms

All

Files

Source is count.c, library is libct.

See Also:

COUNTLEFT()

POSDIFF()

The left-most position there two string differ

Syntax

POSDIFF (<cString1>, <cString2>, [<nIgnore>]) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

POSDIFF() is compatible with CT3's POSDIFF().

Platforms

All

Files

Source is posdiff.c, library is libct.

See Also:

POSEQUAL()

POSEQUAL()

The left-most position there two string begin to be equal

Syntax

POSEQUAL (<cString1>, <cString2>, [<nCompare>], [<nIgnore>]) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

POSEQUAL() is compatible with CT3's POSEQUAL().

Platforms

All

Files

Source is posdiff.c, library is libct.

See Also:

POSDIFF()

FLOOR()

Rounds down a number to the next integer

Syntax

```
FLOOR (nNumber) -> nDownRoundedNumber
```

Arguments

<nNumber>	number to round down

Returns

<nDownRoundedNumber>	the rounded number

Description

The function FLOOR() determines the biggest integer that is smaller than <nNumber>.

Examples

```
? floor (1.1)   --> 1.0
? floor (-1.1)  --> -2.0
```

Tests

```
floor (1.1)  == 1.0
floor (-1.1) == -2.0
```

Status

Ready

Compliance

FLOOR() is compatible with CT3's FLOOR().

Platforms

All

Files

Source is math.c, library is libct.

See Also:

ARRAY()

CEILING()

Rounds up a number to the next integer

Syntax

CEILING (nNumber) -> nUpRoundedNumber

Arguments

<nNumber>	number to round up

Returns

<nUpRoundedNumber>	the rounded number

Description

The function CEILING() determines the smallest integer that is bigger than <nNumber>.

Examples

```
? ceiling (1.1)   --> 2.0
? ceiling (-1.1)  --> -1.0
```

Tests

```
ceiling (1.1)  == 2.0
ceiling (-1.1) == -1.0
```

Status

Ready

Compliance

CEILING() is compatible with CT3's CEILING().

Platforms

All

Files

Source is math.c, library is libct.

See Also:

ARRAY()

SIGN()
Sign of a number

Syntax

SIGN (nNumber) -> nSign

Arguments

<nNumber>	a number

Returns

<nSign>	sign of <nNumber>

Description

The function SIGN() determines the sign of <nNumber>. If <nNumber> is > 0, then SIGN(<nNumber>) returns 1 If <nNumber> is < 0, then SIGN(<nNumber>) returns -1 If <nNumber> is == 0, then SIGN(<nNumber>) returns 0

Examples

```
? sign (1.1)   --> 1
? sign (-1.1)  --> -1
? sign (0.0)  --> 0
```

Tests

```
sign (1.1)  == 1
sign (-1.1) == -1
sign (0.0)  == 0
```

Status

Ready

Compliance

SIGN() is compatible with CT3's SIGN().

Platforms

All

Files

Source is math.c, library is libct.

LOG10()

Decadic logarithm of a number

Syntax

LOG10 (nNumber) -> nLogarithm

Arguments

<nNumber>	number to logarithm

Returns

<nLogarithm>	decadic logarithm of <nNumber>

Description

The function LOG10() calculates the decadic logarithm of <nNumber>, i.e. 10^<nLogarithm> == <nNumber>.

Examples

```
? log10 (10.0)       --> 1.0
? log10 (sqrt(10.0)) --> 0.5
```

Tests

```
log10 (10.0)       == 1.0
log10 (sqrt(10.0)) == 0.5
```

Status

Ready

Compliance

LOG10() is compatible with CT3's LOG10().

Platforms

All

Files

Source is math.c, library is libct.

FACT()

Calculates faculty

Syntax

FACT (nNumber) -> nFaculty

Arguments

<nNumber>	number between 0 and 21

Returns

<nFaculty>	the faculty of <nNumber>

Description

The function FACT() calculates the faculty to the integer given in <nNumber>. The faculty is defined as n! = 1*2*...*n and is often used in statistics. Note, that faculties above 21 are too big so that the function must return a -1.

Examples

```
? fact (0)  --> 1
? fact (1)  --> 1
? fact (4)  --> 24
```

Tests

```
fact (0) == 1
fact (1) == 1
fact (4) == 24
```

Status

Ready

Compliance

FACT() is compatible with CT3's FACT().

Platforms

All

Files

Source is math.c, library is libct.

CELSIUS()

Temperature conversion Fahrenheit to Celsius

Syntax

CELSIUS (nDegreeFahrenheit) --> nDegreeCelsius

Arguments

<nDegreeFahrenheit>	temperature in degree Fahrenheit

Returns

<nDegreeCelsius>	temperate in degree Celsius

Description

CELSIUS() converts temperature values measured in the Fahrenheit scale to the Celsius scale.

Examples

```
// melting point of water in standard conditions
? celsius (32.0)       --> 0.0
// boiling point of water in standard conditions
? celsius (212.0)      --> 100.0
```

Tests

```
celsius (32.0)  == 0.0
celsius (212.0) == 100.0
```

Status

Ready

Compliance

CELSIUS() is compatible with CT3's CELSIUS().

Platforms

All

Files

Source is num1.c, library is libct.

See Also:

FAHRENHEIT()

FAHRENHEIT()

Temperature conversion Celsius to Fahrenheit

Syntax

```
FAHRENHEIT (nDegreeCelsius) --> nDegreeFahrenheit
```

Arguments

<nDegreeCelsius>	temperate in degree Celsius

Returns

<nDegreeFahrenheit>	temperature in degree Fahrenheit

Description

FAHRENHEIT() converts temperature values measured in the Celsius scale to the Fahrenheit scale.

Examples

```
// melting point of water in standard conditions
? fahrenheit (0.0)       --> 32.0
// boiling point of water in standard conditions
? fahrenheit (100.0)      --> 212.0
```

Tests

```
fahrenheit (0.0) == 32.0
celsius (100.0) == 212.0
```

Status

Ready

Compliance

FAHRENHEIT() is compatible with CT3's FAHRENHEIT().

Platforms

All

Files

Source is num1.c, library is libct.

See Also:

CELSIUS()

INFINITY()

Returns the largest floating point number available in the system

Syntax

INFINITY ([<lPlatformIndependant>]) --> nLargestNumber

Arguments

[<lPlatformIndependant>] .T., if the function should return the maximum floating point value available (DBL_MAX) .F., function should try to return the same value as the original CT3 lib did Default: .F.

Returns

<nLargestNumber> the largest floating point number available in the system

Description

INFINITY() returns the largest floating point number available in the system. For platform independance, this is set to DBL_MAX.

Status

Ready

Compliance

INFINITY() must not necessarily return the same number as CT3's INFINITY().

Platforms

All

Files

Source is num1.c, library is libct.

POSALPHA()

Left-most position of a letter in a string

Syntax

POSALPHA (<cString>, [<lMode>], [<nIgnore>]) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

POSALPHA() is compatible with CT3's POSALPHA().

Platforms

All

Files

Source is pos1.c, library is libct.

See Also:

POSLOWER()
POSUPPER()
POSRANGE()

POSLOWER()

Left-most position of a lowercase letter in a string

Syntax

POSLOWER (<cString>, [<lMode>], [<nIgnore>]) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

POSLOWER() is compatible with CT3's POSLOWER().

Platforms

All

Files

Source is pos1.c, library is libct.

See Also:

POSALPHA()
POSUPPER()
POSRANGE()

POSRANGE()

Left-most position of a character from a set in a string

Syntax

```
POSRANGE (<cChar1>, <cChar2>, <cString>, [<lMode>],
[<nIgnore>]) -> nPosition
```

Description

TODO: add documentation

Status

Started

Compliance

POSRANGE() is compatible with CT3's POSRANGE().

Platforms

All

Files

Source is pos1.c, library is libct.

See Also:

POSALPHA()
POSLOWER()
POSUPPER()

POSUPPER()

Left-most position of an uppercase letter in a string

Syntax

POSUPPER (<cString>, [<lMode>], [<nIgnore>]) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

POSUPPER() is compatible with CT3's POSUPPER().

Platforms

All

Files

Source is pos1.c, library is libct.

See Also:

POSALPHA()
POSLOWER()
POSRANGE()

POSCHAR()

Replace character at a certain position within a string

Syntax

```
POSCHAR (<[@]cString>, <cCharacter|nCharacter>, [<nPosition>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

POSCHAR() is compatible with CT3's POSCHAR().

Platforms

All

Files

Source is pos2.c, library is libct.

See Also:

- POSDEL()
- POSINS()
- POSREPL()
- CSETREF()

POSDEL()

Delete characters at a certain position within a string

Syntax

```
POSDEL (<cString>, [<nStartPosition>], <nLength>) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

POSDEL() is compatible with CT3's POSDEL().

Platforms

All

Files

Source is pos2.c, library is libct.

See Also:

POSCHAR()
POSINS()
POSREPL()

POSINS()

Insert characters at a certain position within a string

Syntax

```
POSINS (<cString>, <cInsert>, [<nPosition>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

POSINS() is compatible with CT3's POSINS().

Platforms

All

Files

Source is pos2.c, library is libct.

See Also:

- ARRAY()
- POSDEL()
- POSREPL()

POSREPL()

Replace characters at a certain position within a string

Syntax

POSREPL (<[@]cString>, <cReplacement>, [<nStartPosition>]) -> cString

Description

TODO: add documentation

Status

Started

Compliance

POSREPL() is compatible with CT3's POSREPL().

Platforms

All

Files

Source is pos2.c, library is libct.

See Also:

POSCHAR()
POSDEL()
POSINS()
CSETREF()

CHARRELA()
Character relation of two strings

Syntax

CHARRELA (<cStringToMatch1>, <cString1>,
<cStringToMatch2>, <cString2>) -> nPosition

Description

TODO: add documentation

Status

Started

Compliance

CHARRELA() is compatible with CT3's CHARRELA().

Platforms

All

Files

Source is relation.c, library is libct.

See Also:

CHARRELREP()

CHARRELREP()

Relation dependant character replacement

Syntax

```
CHARRELREP (<cStringToMatch1>, <cString1>,
<cStringToMatch2>, <[@]cString2>,
<cReplacement>) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

CHARRELREP() is compatible with CT3's CHARRELREP().

Platforms

All

Files

Source is relation.c, library is libct.

See Also:

CHARRELA()
CSETREF()

PI()

Returns Pi, the perimeter-to-diameter-ratio of a circle

Syntax

PI () -> nPi

Returns

<nPi> the math constant Pi with maximum precision available

Description

The function PI() can be used if the constant Pi is needed with maximum precision. One of the most known interpretations of this number is the constant perimeter-to-diameter-ratio of circles.

Examples

```
// the diameter of a circle-like swimming pool is 3.4 meters, how
// long is the perimeter ?

? str(PI()*3.4,5,3)+" meters"   --> 10.681 meters
```

Status

Ready

Compliance

PI() is compatible with CT3's PI().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()

SIN()

Sine of the argument

Syntax

SIN (nRadiant) -> nSine

Arguments

<nRadiant>	an angle size given in radiants

Returns

<nSine>	the sine of <nRadiant>

Description

The function SIN() calculates the sine of an angle whose size is given in radiants (full angle equals 2*Pi - see DTOR() for angle size given in degress). A common geometric interpretation of the SIN() function is the counterkathede-hypotenuse-ratio of a right-angled triangle.

Examples

```
? sin (0.0) --> 0.0
? sin (1.0) --> 0.8414...
```

Tests

```
sin (0.0) == 0.0
sin (PI()/4) == sqrt(1/2)
sin (PI()/2) == 1.0
sin (PI()) == 0.0
```

Status

Ready

Compliance

SIN() is compatible with CT3's SIN().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

COS()

Cosine of the argument

Syntax

```
COS (nRadiant) -> nCosine
```

Arguments

<nRadiant>	an angle size given in radiants

Returns

<nCosine>	the cosine of <nRadiant>

Description

The function COS() calculates the cosine of an angle whose size is given in radiants (full angle equals 2*Pi - see DTOR() for angle size given in degress). A common geometric interpretation of the COS() function is the ankathede-hypotenuse-ratio of a right-angled triangle.

Examples

```
? cos (0.0) --> 1.0
? cos (1.0) --> 0.5403...
```

Tests

```
cos (0.0) == 1.0
cos (PI()/4) == sqrt(1/2)
cos (PI()/2) == 0.0
cos (PI()) == -1.0
```

Status

Ready

Compliance

COS() is compatible with CT3's COS().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

TAN()

Tangent of the argument

Syntax

TAN (nRadiant) -> nTangent

Arguments

<nRadiant>	an angle size given in radiants

Returns

<nTangent>	the tangent of <nRadiant>

Description

The function TAN() calculates the tangent of an angle whose size is given in radiants (full angle equals 2*Pi - see DTOR() for angle size given in degress). A common geometric interpretation of the TAN() function is the counterkathede-ankathede-ratio of a right-angled triangle, or, tan(x) = sin(x)/cos(x).

Examples

```
? tan (0.0) --> 0.0
? tan (1.0) --> 1.5574...
```

Tests

```
tan (0.0) == 0.0
tan (PI()/4) == 1
tan (PI()) == 0.0
```

Status

Ready

Compliance

TAN() is compatible with CT3's TAN().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

COT()

Cotangent of the argument

Syntax

COT (nRadiant) -> nCotangent

Arguments

<nRadiant>	an angle size given in radiants

Returns

<nCotangent>	the cotangent of <nRadiant>

Description

The function COT() calculates the cotangent of an angle whose size is given in radiants (full angle equals 2*Pi - see DTOR() for angle size given in degress). A common geometric interpretation of the COT() function is the ankathede-counterkathede-ratio of a right-angled triangle, or, cot(x) = cos(x)/sin(x)=1/tan(x).

Examples

```
? cot (1.0) --> 0.6420...
```

Tests

```
cot (PI()/4) == 1
cot (PI()/2) == 0
```

Status

Ready

Compliance

COT() is compatible with CT3's COT().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

ASIN()

Arcus sine of the argument

Syntax

```
ASIN (nSine) -> nRadiant
```

Arguments

<nSine>	the sine of an angle

Returns

<nRadiant>	the angle whose sine is <nSine>

Description

The function ASIN() is the inverse function of SIN(). It takes a sine value and returns the smallest(!) angle whose sine equals to the argument. The return value is given in radiants (full angle equals 2*Pi - see DTOR() if you need to convert it into degress). Note, that <nSine> must be between -1 and 1 and that <nRadiant> is always between -PI()/2 and PI()/2.

Examples

```
? asin (0.0) --> 0.0
? asin (0.5) --> 0.5235...
```

Tests

```
asin (0.0) == 0.0
asin (sqrt(1/2)) == PI()/4
asin (1.0) == PI()/2
asin (0.0) == 0.0  // and not PI(), since the smallest angle is returned !
```

Status

Ready

Compliance

ASIN() is compatible with CT3's ASIN().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

ACOS()

Arcus cosine of the argument

Syntax

```
ACOS (nCosine) -> nRadiant
```

Arguments

<nCosine>	the cosine of an angle

Returns

<nRadiant>	the angle whose cosine is <nCosine>

Description

The function ACOS() is the inverse function of COS(). It takes a cosine value and returns the smallest(!) angle whose cosine equals to the argument. The return value is given in radiants (full angle equals 2*Pi - see DTOR() if you need to convert it into degress). Note, that <nCosine> must be between -1 and 1 and that <nRadiant> is always between 0 and PI().

Examples

```
? acos (0.0) --> PI()/2
? acos (0.5) --> 1.04719...
```

Tests

```
acos (0.0) == PI()/2
acos (sqrt(1/2)) == PI()/4
acos (1.0) == 0.0
acos (-1.0) == PI()
acos (0.0) == PI()/2  // and not -PI()/2, although cos (-PI()/2) == 0.0 !
```

Status

Ready

Compliance

ACOS() is compatible with CT3's ACOS().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

ATAN()
Arcus tangent of the argument

Syntax

```
ACOS (nTangent) -> nRadiant
```

Arguments

<nTangent>	the tangent of an angle

Returns

<nRadiant>	the angle whose tangent is <nTangent>

Description

The function ATAN() is the inverse function of TAN(). It takes a tangent value and returns the smallest(!) angle whose tangent equals to the argument. The return value is given in radiants between -PI()/2 and PI()/2 (full angle equals 2*Pi - see DTOR() if you need to convert it into degress).

Examples

```
? atan (0.0) --> 0.0
? atan (0.5) --> 0.4636...
```

Tests

```
atan (0.0) == 0.0
atan (1.0) == PI()/4
atan (0.0) == 0.0 // and not PI(), although tan (PI()) == 0.0 !
```

Status

Ready

Compliance

ATAN() is compatible with CT3's ATAN().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

ATN2()

Arcus tangent a sine and a cosine argument

Syntax

ATN2 (nSine, nCosine) -> nRadiant

Arguments

<nSine> the sine of an angle <nCosine> the cosine of an angle

Returns

<nRadiant> the angle whose tangent is <nSine>/<nCosine>

Description

The function ATN2() is an alternate function for calculating the arcus tangent, atn2(x,y) = atan(x/y). It takes two arguments, the sine and the cosine of the angle that should be calculated. Thus, in contrast to the ATAN() function, ATN2() can distinguish whether the sine or the cosine has a negative sign (or both being positive or negative), so that the return value can be between -PI() and PI() and covers the full angle. The return value is given in radiants (full angle equals 2*Pi - see DTOR() if you need to convert it into degress).

Examples

```
? atn2 (0.0, 1.0) --> 0.0
? atn2 (sqrt(1/2), sqrt(1/2)) --> PI()/4
```

Tests

```
atn2 (0.0, 1.0) == 0.0
atn2 (sqrt(1/2),sqrt(1/2)) == PI()/4
atn2 (-sqrt(1/2),-sqrt(1/2)) == -3/4*PI()  // atan() would return PI()/4 !
```

Status

Ready

Compliance

ATN2() is compatible with CT3's ATN2().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
SINH()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

SINH()

Hyperbolic Sine of the argument

Syntax

SINH (nArea) -> nHyperbolicSine

Arguments

<nArea>	the size of the area (see below)

Returns

<nHyperbolicSine>	the hyperbolic sine of <nArea>

Description

The function SINH() calculates the hyperbolic sine of the argument. In analytical mathematics it is defined as 1/2*(exp(nArea)-exp(-nArea)). A common geometric interpretation of the SINH() function is the maximum y value of the points in the area with the given size <nArea>, that is bound by the x axis, a straight line through the point of origin (this one is fixed by the area) and the hyperbola x^2-y^2=1.

Examples

```
? sinh (0.0) --> 0.0
? sinh (1.0) --> 1.1752...
```

Tests

```
sinh (0.0) == 0.0
sinh (-0.5) == -sinh(0.5)
```

Status

Ready

Compliance

SINH() is new in Harbours CT3's library.

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
COSH()
ARRAY()
RTOD()
DTOR()
PI()

COSH()

Hyperbolic Cosine of the argument

Syntax

COSH (nArea) -> nHyperbolicCosine

Arguments

<nArea>	the size of the area (see below)

Returns

<nHyperbolicCosine>	the hyperbolic cosine of <nArea>

Description

The function COSH() calculates the hyperbolic cosine of the argument. In analytical mathematics it is defined as 1/2*(exp(nArea)+exp(-nArea)). A common geometric interpretation of the COSH() function is the maximum x value of the points in the area with the given size <nArea>, that is bound by the x axis, a straight line through the point of origin (this one is fixed by the area) and the hyperbola x^2-y^2=1.

Examples

```
? cosh (0.0) --> 1.0
? cosh (1.0) --> 1.5430...
```

Tests

```
cosh (0.0) == 1.0
cosh (-0.5) == cosh(0.5)
```

Status

Ready

Compliance

COSH() is new in Harbours CT3's library.

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
ARRAY()
RTOD()
DTOR()
PI()

SINH()

Hyperbolic Tangent of the argument

Syntax

TANH (nArea) -> nHyperbolicTangent

Arguments

<nArea>	the size of the area (see below)

Returns

<nHyperbolicTangent>	the hyperbolic tangent of <nArea>

Description

The function TANH() calculates the hyperbolic tangent of the argument. In analytical mathematics it is defined as SINH(x)/COSH(x).

Examples

```
? tanh (0.0) --> 0.0
? tanh (1.0) --> 0.7615...
```

Tests

```
tanh (0.0) == 0.0
tanh (-0.5) == -tanh(0.5)
```

Status

Ready

Compliance

TANH() is new in Harbours CT3's library.

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
RTOD()
DTOR()
PI()

RTOD()

Convert radiant to degree

Syntax

RTOD (nRadiant) -> nDegree

Arguments

<nRadiant>	the size of an angle in radiant

Returns

<nDegree>	the size of that angle in degree

Description

The function RTOD() can be used to convert sizes of angles given in radiant (like those returned by the asin, acos or atan function) to degrees that are commonly used geometry and technics.

Examples

```
? rtod (PI()) --> 180
? tanh (PI()/3) --> 60
```

Tests

```
rtod (0.0) == 0.0
rtod (PI()) == 180.0
```

Status

Ready

Compliance

RTOD() is compatible with CT3's RTOD().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
DTOR()
PI()

DTOR()

Convert degree to radiant

Syntax

DTOR (nDegree) -> nRadiant

Arguments

<nDegree>	the size of that angle in degree

Returns

<nRadiant>	the size of an angle in radiant

Description

The function DTOR() can be used to convert sizes of angles given in degrees to radiant (as expected by sin, cos or tan functions).

Examples

```
? dtor (180) --> PI()
? dtor (60) --> PI()/3
```

Tests

```
dtor (0.0) == 0.0
dtor (180.0) == PI()
```

Status

Ready

Compliance

DTOR() is compatible with CT3's DTOR().

Platforms

All

Files

Source is trig.c, library is libct.

See Also:

SIN()
COS()
TAN()
COT()
ASIN()
ACOS()
ATAN()
ATN2()
SINH()
COSH()
ARRAY()
RTOD()
PI()

NUMAND()

Bitwise logical AND operation on 16-bit numbers

Syntax

```
NUMAND( <nWORD1|cHexWORD1>, <nWORD2|cHexWORD2>[, ..<nWORDn|cHexWORDn>)
-> <nWordAND>
```

Description

TODO: add documentation

Status

Started

Compliance

NUMAND() is compatible with CT3's NUMAND().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

NUMOR()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMOR()

Bitwise logical OR operation on 16-bit numbers

Syntax

```
NUMOR( <nWORD1|cHexWORD>1, <nWORD2|cHexWORD2>[, ..<nWORDn|cHexWORDn>)
-> <nWordOR>
```

Description

TODO: add documentation

Status

Started

Compliance

NUMOR() is compatible with CT3's NUMOR().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

NUMAND()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMXOR()

Bitwise logical XOR operation on 16-bit numbers

Syntax

NUMXOR(<nWORD1|cHexWORD1>, <nWORD2|cHexWORD2>) -> <nWordXOR>

Description

TODO: add documentation

Status

Started

Compliance

NUMXOR() is compatible with CT3's NUMXOR().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

NUMAND()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMNOT()

Bitwise logical NOT operation on 16-bit numbers

Syntax

NUMNOT(<nWORD|cHexWORD>) -> <nWordNOT>

Description

TODO: add documentation

Status

Started

Compliance

NUMNOT() is compatible with CT3's NUMNOT().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

NUMAND()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMROL()

Bitwise ROL (rotate left) operation on 16-bit numbers

Syntax

```
NUMROL( <nWORD1|cHexWORD1>, <nWORD1|cHexWORD1>[, <lLowByte>] )
-> <nWordROL>
```

Description

TODO: add documentation

Status

Started

Compliance

NUMROL() is compatible with CT3's NUMROL().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

NUMAND()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMMIRR()

Bitwise mirror operation on 8-bit and 16-bit numbers

Syntax

```
NUMMIRR( <nNumber|cHexNum>[, <l8/16bit>] ) -> <nResult>
```

Description

TODO: add documentation

Status

Started

Compliance

NUMMIRR() is compatible with CT3's NUMMIRR().

Platforms

All

Files

Source is bit1.c, library is libct.

See Also:

- NUMAND()
- ARRAY()
- ARRAY()
- ARRAY()
- ARRAY()

NUMAT()

Number of occurrences of a sequence in a string

Syntax

NUMAT (<cStringToMatch>, <cString>, [<nIgnore>]) --> nCount

Description

TODO: add documentation

Status

Started

Compliance

NUMAT() is compatible with CT3's NUMAT().

Platforms

All

Files

Source is numat.c, library is libct.

See Also:

CSETATMUPA()
SETATLIKE()

PADLEFT()

Fills string to a certain length on the left

Syntax

PADLEFT (<cString>,<nLength>, [<cChar|nChar>]) -> cString

Description

TODO: add documentation

Status

Started

Compliance

PADLEFT() is compatible with CT3's PADLEFT().

Platforms

All

Files

Source is pad.c, library is libct.

See Also:

PADRIGHT()

PADRIGHT()

Fills string to a certain length on the right

Syntax

```
PADRIGHT (<cString>,<nLength>, [<cChar|nChar>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

PADRIGHT() is compatible with CT3's PADRIGHT().

Platforms

All

Files

Source is pad.c, library is libct.

See Also:

PADLEFT()

RANGEREM()

Remove characters within a certain ASCII range from a string

Syntax

```
RANGEREM (<cChar1|nChar1>, <cChar2|nChar2>, <cString>) -> cString
```

Description

TODO: add documentation

Examples

```
? rangerem ("0","9","year2002.dbf") // "year.dbf", remove all digits
? rangerem ("9","0","year2002.dbf") // "22", testing removal from "9" to chr(255)
                                    // and from chr(0) to "0"
? rangerem ("0","9","yearcurr.dbf") // "yearcurr.dbf", test leaving string untouched
```

Tests

```
rangerem ("0","9","year2002.dbf") == "year.dbf"
rangerem ("9","0","year2002.dbf") == "22"
rangerem ("0","9","yearcurr.dbf") == "yearcurr.dbf"
```

Status

Started

Compliance

RANGEREM() is compatible with CT3's RANGEREM().

Platforms

All

Files

Source is range.c, library is libct.

See Also:

ARRAY()

RANGEREPL

Replace characters within a certain ASCII range from a string

Syntax

```
RANGEREPL (<cChar1|nChar1>, <cChar2|nChar2>,
<[@]cString>, <cReplacementChar|nReplacementChar>) -> cString
```

Description

TODO: add documentation

Examples

```
? rangerepl ("0","9","year2002.dbf","?") // "year????.dbf", replace all digits
? rangerepl ("9","0","year2002.dbf","?") // "????2??2????", testing replacement from "9"
                                         // and from chr(0) to "0"
? rangerepl ("0","9","yearcurr.dbf","?") // "yearcurr.dbf", test leaving string untouched
```

Tests

```
rangerepl ("0","9","year2002.dbf","?") == "year????.dbf"
rangerepl ("9","0","year2002.dbf","?") == "????2??2????"
rangerepl ("0","9","yearcurr.dbf","?") == "yearcurr.dbf"
```

Status

Started

Compliance

RANGEREPL() is compatible with CT3's RANGEREPL().

Platforms

All

Files

Source is range.c, library is libct.

See Also:

RANGEREM()

REMALL()

Remove certain characters at the left and right of a string

Syntax

REMALL (<cString>, [<cSearch|nSearch>]) -> cString

Description

TODO: add documentation

Status

Started

Compliance

REMALL() is compatible with CT3's REMALL().

Platforms

All

Files

Source is remove.c, library is libct.

See Also:

REMLEFT()
REMRIGHT()

REMLEFT()

Remove certain characters at the left of a string

Syntax

REMLEFT (<cString>, [<cSearch|nSearch>]) -> cString

Description

TODO: add documentation

Status

Started

Compliance

REMLEFT() is compatible with CT3's REMLEFT().

Platforms

All

Files

Source is remove.c, library is libct.

See Also:

REMALL()
REMRIGHT()

REMRIGHT()

Remove certain characters at the right of a string

Syntax

```
REMRIGHT (<cString>, [<cSearch|nSearch>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

REMRIGHT() is compatible with CT3's REMRIGHT().

Platforms

All

Files

Source is remove.c, library is libct.

See Also:

REMALL()
REMLEFT()

REPLALL()

Replace certain characters at the left and right of a string

Syntax

```
REPLALL (<cString>, <cReplace|nReplace>, [<cSearch|nSearch>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

REPLALL() is compatible with CT3's REPLALL().

Platforms

All

Files

Source is replace.c, library is libct.

See Also:

REPLLEFT()
REPLRIGHT()

REPLLEFT()

Replace certain characters at the left of a string

Syntax

```
REPLLEFT (<cString>, <cReplace|nReplace>, [<cSearch|nSearch>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

REPLLEFT() is compatible with CT3's REPLLEFT().

Platforms

All

Files

Source is replace.c, library is libct.

See Also:

REPLALL()
REPLRIGHT()

REPLRIGHT()

Replace certain characters at the right of a string

Syntax

```
REPLRIGHT (<cString>, <cReplace|nReplace>, [<cSearch|nSearch>]) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

REPLRIGHT() is compatible with CT3's REPLRIGHT().

Platforms

All

Files

Source is replace.c, library is libct.

See Also:

REPLALL()
REPLLEFT()

STRSWAP()

Swap the contents of two strings

Syntax

```
STRSWAP (<[@]cString1>, <[@]cString2>) -> cString
```

Description

TODO: add documentation

Status

Started

Compliance

STRSWAP() is compatible with CT3's STRSWAP().

Platforms

All

Files

Source is strswap.c, library is libct.

WORDTOCHAR()

Replace double with single characters

Syntax

WORDTOCHAR (<cDoubleCharacterSearchString>, <cString>, <cSingleCharacterReplaceString>) -> cString

Description

TODO: add documentation

Status

Started

Compliance

WORDTOCHAR() is compatible with CT3's WORDTOCHAR().

Platforms

All

Files

Source is wordtoch.c, library is libct.

See Also:

CSETATMUPA()
CHARREPL()
WORDREPL()

CSETARGERR()

Sets argument error behaviour

Syntax

CSETARGERR ([<nNewMode>]) -> <nOldMode>

Arguments

[<nNewMode>]	New argument error throwing mode

Returns

<nOldMode>	The current or old argument error throwing mode.

Description

All CT3 functions are very compliant in their reaction to wrong parameters. By using the CSETARGERR() function, you can make the library throw an error with the severity <nNewMode>. It is then up to the error handler to substitute the return value. <nNewMode> can be one of the severity modes defined in ct.ch: CT_ARGERR_WHOCARES   corresponds to ES_WHOCARES CT_ARGERR_WARNING corresponds to ES_WARNING CT_ARGERR_ERROR   corresponds to ES_ERROR CT_ARGERR_CATASTROPHIC corresponds to ES_CATASTROPHIC CT_ARGERR_IGNORE The last is the default behaviour and switches any argument error throwing off.

Status

Ready

Compliance

CSETARGERR() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ct.c, library is libct.

CTCINIT()

Initializes the CT3 library, C part

Syntax

CTCINIT () -> lInitialized

Arguments

Returns

Description

The CTCINIT() function initializes the C source part of the CT3 library. Do not call this function directly.

Status

Ready

Compliance

CTCINIT() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ctc.c, library is libct.

See Also:

CTINIT()
CTEXIT()

CTCEXIT()

Uninitializes the CT3 library, C part

Syntax

```
CTCEXIT () -> nil
```

Arguments

Returns

Description

The CTCEXIT() function uninitializes the C part of the CT3 library. Do not call this function directly.

Status

Ready

Compliance

CTCEXIT() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ctc.c, library is libct.

See Also:

CTINIT()
CTEXIT()

KSETINS()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is keyset.c, library is libct.

KSETCAPS()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is keyset.c, library is libct.

KSETNUM()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is keyset.c, library is libct.

KSETSCROLL()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is keyset.c, library is libct.

PRINTSTAT()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is print.c, library is libct.

PRINTREADY()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

DOS

Files

Source is print.c, library is libct.

CLEARBIT()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is bit2.c, library is libct.

See Also:

SETBIT()
ARRAY()

SETBIT()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is bit2.c, library is libct.

See Also:

CLEARBIT()
ARRAY()

ISBIT()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is bit2.c, library is libct.

See Also:

CLEARBIT()
ARRAY()

NUMANDX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

Returns

NUMANDX() join all designated parameters with the logical "AND" and return the result.

Description

This function is similar to NUMAND() function with a significative change. The first parameter indicate the quantity of lower bits of nLONG are used. If MSB of the result is ON the number is considerate a negative number. In other words, if <nSignificativeBits> = 16, nResult return a number between -32768 and 32767; if <nSignificativeBits> = 8, nResult return a number between -128 and 127.

TODO: add documentation

Status

Started

Compliance

NUMANDX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMAND()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMORX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

Returns

NUMORX() join all designated parameters with the logical "OR" and return the result.

Description

This function is similar to NUMOR() function with a significative change. The first parameter indicate the quantity of lower bits of nLONG are used. If MSB of the result is ON the number is considerate a negative number. In other words, if <nSignificativeBits> = 16, nResult return a number between -32768 and 32767; if <nSignificativeBits> = 8, nResult return a number between -128 and 127.

TODO: add documentation

Status

Started

Compliance

NUMORX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMOR()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMXORX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

Returns

NUMXORX() join all designated parameters with the logical "XOR" and return the result.

Description

This function is similar to NUMXOR() function with a significative change. The first parameter indicate the quantity of lower bits of nLONG are used. If MSB of the result is ON the number is considerate a negative number. In other words, if <nSignificativeBits> = 16, nResult return a number between -32768 and 32767; if <nSignificativeBits> = 8, nResult return a number between -128 and 127.

TODO: add documentation

Status

Started

Compliance

NUMXORX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMXOR()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMNOTX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

Returns

NUMNOTX() return the negated binary value of the nLONG parameter. The 0 bits become 1, and 1 bits become 0.

Description

This function is similar to NUMNOT() function with a significative change.
The first parameter indicate the quantity of lower bits of nLONG are used. If MSB of the result is ON the number is considerate a negative number. In other words, if <nSignificativeBits> = 16, nResult return a number between -32768 and 32767; if <nSignificativeBits> = 8, nResult return a number between -128 and 127.

TODO: add documentation

Status

Started

Compliance

NUMNOTX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMNOT()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMROLX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

<nWORD | cHexWORD> Designate a number of rotations in the range of 1 to <nSignificativeBits>; as either numeric or hexadecimal.

Returns

NUMROLX() return the rotation result.

Description

This function is similar to NUMROL() function with a significative change. The first parameter indicate the quantity of lower bits of nLONG are used. When the high bit rotates it is not just moved out to the left, it is also moved in on the right. The not rotated bits is not moved.

TODO: add documentation

Status

Started

Compliance

NUMROLX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMROL()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

NUMMIRRX()

$SYNTAX$

Arguments

<SignificativeBits> Designate a number in the range of 0 to 32, indicating the LSB of nLONGx|cHexLONGx that will be used.

<nLONG | cHexLONG> Designate either decimal or hexadecimal number string.

Returns

NUMMIRR() returns a value by which the bit opposite the first parameter is mirrored.

Description

This function is similar to NUMMIRR() function with a significative change. The first parameter indicate the quantity of lower bits of nLONG are used. When you mirror bit, bit 1 interchanges with bit <nSignificativeBits>, bit 2 with bit <nSignificativeBits> - 1, etc.. The not mirrored bits is not moved.

TODO: add documentation

Status

Started

Compliance

NUMMIRRX() is a new function in the CT3-library for Harbour.

Platforms

All

Files

Source is bit3.c, library is libct.

See Also:

NUMMIRR()
ARRAY()
ARRAY()
ARRAY()
ARRAY()
ARRAY()

FTOC()

$SYNTAX$

Arguments

<nFloatingPointNumber> Designate any Harbour number.

Returns

FTOC() return a string with the size of DOUBLE. ATTENTION: different implementations or platforms of Harbour, they could produce different format in the string returned by FTOC().

Description

Harbour internal numbers in Floating Point are stored in data type DOUBLE. FTOC() returns these bits as an string. In this way, numbers con be saved more compactly.

TODO: add documentation

Status

Started

Platforms

All

Files

Source is ftoc.c, library is libct.

See Also:

CTOF()
ARRAY()

CTOF()

$SYNTAX$

Arguments

<cFloatingPointNumber> Designate a string that contains a Harbour number in flotaing point format. ATTENTION: different implementations or platforms of Harbour, they could produce different format in the string returned by FTOC().

Returns

CTOF() return the floating point number that corresponds to the string passed.

Description

Character strings created with FTOC() or XTOC() are convert into Harbour floating point number

TODO: add documentation

Status

Started

Platforms

All

Files

Source is ftoc.c, library is libct.

See Also:

FTOC()
ARRAY()

XTOC()

$SYNTAX$

Arguments

<expValue> Designate an expression of some of the following data type: NUMBER, CHARACTER, DATE, LOGICAL.

Returns

XTOC() return a string with the representation of data type of expValue. ATTENTION: different implementations or platforms of Harbour, they could produce different format in the string returned by XTOC() for data type NUMBER.

Description

Each data type always returns a string with a particular fixed length:

```
----------------------------------------------------------- Data Type
Result Length      Similar function
----------------------------------------------------------- Numeric      sizeof(
DOUBLE )   FTOC()  Logical      1  Date         8                 DTOS()  String
Unchanged  -----------------------------------------------------------
```

TODO: add documentation

Status

Started

Platforms

All

Files

Source is misc1.c, library is libct.

See Also:

CTOF()
ARRAY()

TABEXPAND()

Replace tabulator control characters with fill characters

Syntax

```
TABEXPAND (<cString>, [<nTabWidth>], [<cFillChar|nFillChar>],
[<cNewLineCharacters>], [<cTabChar|nTabChar>],
[<lIgnore141>]) -> cExpandedString
```

Arguments

```
indicating new line,  default is the string returned by  hb_osnewline()
<cTabChar|nTabChar>       character indicating a tab stop,  default is chr(9)
<lIgnore141>              .T., if the soft-CR used by MEMOEDIT()  should be ignored
as a newline indicator,  default is .F. (functions uses chr(141))
```

Description

TODO: add documentation

Tests

```
tabexpand("-"+chr(9)+"!")             == "-       !"
tabexpand("----"+chr(9) +"!")         == "----    !"
tabexpand("-"+chr(9)+"!",, "+")       == "-+++++++!"
tabexpand("-"+chr(9)+ "!", 4)         == "-   !"
tabexpand("----"+chr(9)+ "!", 8)      == "----    !"
tabexpand("----"+chr(9)+ "!", 8, "+") == "----++++!"
tabexpand("-"+chr(9)+"!"+hb_osnewline()+"----"+chr(9)+ "!",, "+") == "-+++++++!"+hb_osnew
```

Status

Started

Compliance

TABEXPAND() is compatible with CT3's TABEXPAND(), but there are three new parameters for a better fine control of the function's behaviour.

Platforms

All

Files

Source is tab.c, library is libct.

See Also:

TABPACK()

TABPACK()

Pack fill characters to appropriate tab characters

Syntax

```
TABPACK (<cString>, [<nTabWidth>], [<cFillChar|nFillChar>],
[<cNewLineCharacters>], [<cTabChar|nTabChar>],
[<lIgnore141>]) -> cPackedString
```

Arguments

```
indicating new line,  default is the string returned by  hb_osnewline()
<cTabChar|nTabChar>       character indicating a tab stop,  default is chr(9)
<lIgnore141>              .T., if the soft-CR used by MEMOEDIT()  should be ignored
as a newline indicator,  default is .F. (functions uses chr(141))
```

Description

TODO: add documentation

Status

Started

Compliance

TABPACK() is compatible with CT3's TABPACK(), but there are three new parameters for a better fine control of the function's behaviour.

Platforms

All

Files

Source is tab.c, library is libct.

See Also:

TABEXPAND()

MANTISSA()

Evaluate the mantissa of a floating point number

Syntax

MANTISSA(<nFloatingPointNumber>) --> nMantissa

Arguments

<nFloatingPointNumber> Designate any Harbour number.

Returns

MANTISSA() returns the mantissa of the <nFloatingPointNumber> number.

Description

This function supplements EXPONENT() to return the mantissa of the <nFloatingPointNumber> number.

Note: The mantissa value can be 0 or in the range of 1 to 2.

The following calculation reproduces the original value:

MANTISSA(<nFloatingPointNumber>)* 2^EXPONENT(<nFloatingPointNumber>) = <nFloatingPointNumber>

TODO: add documentation

Status

Started

Compliance

MANTISSA() is compatible with CT3's MANTISSA().

Platforms

All

Files

Source is exponent.c, library is libct.

See Also:

EXPONENT()

EXPONENT()

Evaluate the exponent of a floating point number

Syntax

EXPONENT(<nFloatingPointNumber>) --> nExponent

Arguments

<nFloatingPointNumber> Designate any Harbour number.

Returns

EXPONENT() returns the exponent of the <nFloatingPointNumber> number in base 2.

Description

This function supplements MANTISSA() to return the exponent of the <nFloatingPointNumber> number.

Values > 1 or values < -1 return a positive number 0 to 1023.

Values < 1 or values > -1 return a negative number -1 to -1023.

The EXPONENT(0), return 0.

The following calculation reproduces the original value:

2^EXPONENT(<nFloatingPointNumber>) * MANTISSA(<nFloatingPointNumber>) = <nFloatingPointNumber>

TODO: add documentation

Status

Started

Compliance

EXPONENT() is compatible with CT3's EXPONENT()

Platforms

All

Files

Source is exponent.c, library is libct.

See Also:

MANTISSA()

SCREENATTR()

$SYNTAX$

Arguments

<nRow> Designates the line from which to determine the attribute. The default is the cursor line.

<nColumn> Designates the column from which to determine the attribute. The default is the cursor column.

Returns

SCREENATTR() returns the attribute at the designated position.

Description

SCREENATTR() returns the current screen attribute at <nRow> and <nColumn>. You can query targeted attributes this way and save them to use later, or process them later with INVERTATTR().

TODO: add documentation

Status

Started

Platforms

All

Files

Source is screen1.c, library is libct.

STRDIFF()

Evaluate the "Edit (Levensthein) Distance" of two strings

Syntax

STRDIFF (<cString1>, <cString2>, [<nReplacementPenalty>], [<nDeletionPenalty>], [<nInsertionPenalty>]) -> <nDistance>

Arguments

<cString1> string at the "starting point" of the transformation process, default is "" <cString2> string at the "end point" of the transformation process, default is "" <nReplacementPenalty> penalty points for a replacement of one character, default is 3 <nDeletionPenalty> penalty points for a deletion of one character, default is 6 <nInsertionPenalty> penalty points for an insertion of one character, default is 1

Returns

<nDistance> penalty point sum of all operations needed to transform <cString1> to <cString2>

Description

The STRDIFF() functions calculates the so called "Edit" or "Levensthein" distance of two strings. This distance is a measure for the number of single character replace/insert/delete operations (so called "point mutations") required to transform <cString1> into <cString2> and its value will be the smallest sum of the penalty points of the required operations.

Be aware that this function is both quite time - O(len(cString1)*len(cString2)) - and memory consuming - O((len(cString1)+1)*(len(cString2)+1)*sizeof(int)) - so keep the strings as short as possible. E.g., on common 32 bit systems (sizeof(int) == 4), calling strdiff() with two strings of 1024 bytes in length will consume 4 MB of memory. To not impose unneeded restrictions, the function will only check if (len(cString1)+1)*(len(cString2)+1)*sizeof(int) <= UINT_MAX, although allocing UINT_MAX bytes will not work on most systems. If this simple check fails, -1 is returned.

Also, be aware that there can be an overflow when the penalty points are summed up: Assuming that the number of transformation operations is in the order of max(len(cString1),len(cString2)), the penalty point sum, that is internally stored in an "int" variable, is in the order of
Also, be aware that there can be an overflow when the penalty points are ertionPentaly). The STRDIFF() does not do an overflow check due to time performance reasons. Future versions of STRDIFF() could use a type different to "int" to store the penalty point sum to save memory or to avoid overflows.

The function is aware of the settings done by SETATLIKE(), that means that the wildchar character is considered equal to ALL characters.

```
? strdiff("ABC", "ADC") // 3, one character replaced
? strdiff("ABC", "AEC") // 3, dito
? strdiff("CBA", "ABC") // 6, two characters replaced
? strdiff("ABC", "AXBC") // 1, one character inserted
? strdiff("AXBC", "ABC") // 6, one character removed
? strdiff("AXBC", "ADC") // 9, one character removed and one replaced
```

Tests

```
strdiff("ABC", "ADC") == 3
strdiff("ABC", "AEC") == 3
strdiff("CBA", "ABC") == 6
strdiff("ABC", "AXBC") == 1
strdiff("AXBC", "ABC") == 6
strdiff("AXBC", "ADC") == 9
```

Status

Ready

Compliance

STRDIFF() is compatible with CT3's STRDIFF().

Platforms

All

Files

Source is strdiff.c, library is libct.

See Also:

SETATLIKE()

CTINIT()

Initializes the CT3 library

Syntax

CTINIT () -> lInitialized

Arguments

Returns

Description

The CTINIT() function initializes the CT3 library. Identical code is declared as INIT FUNCTION, thus should be executed automatically at the beginning of the application, but it is a good idea to call it once again explicitly somewhere at the beginning of your program to check the initialization.

Status

Ready

Compliance

CTINIT() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ct.prg, library is libct.

CTEXIT()

Uninitializes the CT3 library

Syntax

CTEXIT () -> nil

Arguments

Returns

Description

The CTEXIT() function uninitializes the CT3 library. Identical code is declared as EXIT FUNCTION, thus should be executed automatically at the end of the application, but it is a good idea to call it explicitly somewhere at the end of your program to make sure that the deinitialization takes place.

Status

Ready

Compliance

CTEXIT() is a new function in Harbour's CT3 library.

Platforms

All

Files

Source is ct.prg, library is libct.

BOM()

_B_egin _O_f _M_onth

Syntax

BOM ([<dDate>]) -> dDateBeginOfMonth

Description

TODO: add documentation

Status

Started

Compliance

BOM() is compatible with CT3's BOM().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

EOM()
BOQ()
EOQ()
BOY()
EOY()

EOM()

_E_nd _O_f _M_onth

Syntax

EOM ([<dDate>]) -> dDateEndOfMonth

Description

TODO: add documentation

Status

Started

Compliance

EOM() is compatible with CT3's EOM().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

BOM()
BOQ()
EOQ()
BOY()
EOY()

BOQ()

_B_egin _O_f _Q_uarter

Syntax

BOQ ([<dDate>]) -> dDateBeginOfQuarter

Description

TODO: add documentation

Status

Started

Compliance

BOQ() is compatible with CT3's BOQ().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

- BOM()
- EOM()
- EOQ()
- BOY()
- EOY()

EOQ()

_E_nd _O_f _Q_uarter

Syntax

EOQ ([<dDate>]) -> dDateEndOfQuarter

Description

TODO: add documentation

Status

Started

Compliance

EOQ() is compatible with CT3's EOQ().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

BOM()
EOM()
BOQ()
BOY()
EOY()

BOY()

_B_egin _O_f _Y_ear

Syntax

BOY ([<dDate>]) -> dDateBeginOfYear

Description

TODO: add documentation

Status

Started

Compliance

BOY() is compatible with CT3's BOY().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

BOM()
EOM()
BOQ()
EOQ()
EOY()

EOY()

_E_nd _O_f _Y_ear

Syntax

EOY ([<dDate>]) -> dDateEndOfYear

Description

TODO: add documentation

Status

Started

Compliance

EOY() is compatible with CT3's EOY().

Platforms

All

Files

Source is datetime.prg, library is libct.

See Also:

BOM()
EOM()
BOQ()
EOQ()
BOY()

STOD()

Convert ANSI date string to Harbour date

Syntax

STOD ([<cDate>]) -> dDate

Description

TODO: add documentation

Status

Started

Compliance

STOD() is compatible with CT3's STOD().

Platforms

All

Files

Source is datetime.prg, library is libct.

NTOC()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is numconv.prg, library is libct.

See Also:

CTON()

CTON()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is numconv.prg, library is libct.

See Also:

NTOC()

BITTOC()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is numconv.prg, library is libct.

See Also:

CTOBIT()

CTOBIT()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is numconv.prg, library is libct.

See Also:

BITTOC()

NTOCOLOR()

$SYNTAX$

Arguments

<nAttr>   Designates the value for the combined numeric color attributes.

<lColorCode>   If designated as .F. or if the parameter is omitted, NTOCOLOR() returns a string with a numeric color code. When designated as .T., NTOCOLOR() returns a string with the CA-Clipper alpha color coding.

Returns

NTOCOLOR() returns the designated color attribute in the NN/NN or CC/CC form.

Description

NTOCOLOR() converts a color attribute returned from another function in numeric form, into the alphanumeric data format. Use this attribute in conjunction with the CA-Clipper SET COLOR TO command.

TODO: add documentation

Status

Started

Platforms

All

Files

Source is color.prg, library is libct.

COLORTON()

$SYNTAX$

Arguments

<cAttr> Designates the alphanumeric color attribute that is converted in NN/NN or CC/CC form.

Returns

COLORTON() returns a number that corresponds to the combined numeric color attribute.

Description

COLOR TO (N)umeric The function changes an alphanumeric color attribute from NN/NN or CC/CC into a combined numeric attribute. These combined attribute values are useful with the CA-Clipper Tools functions STRSCREEN(), SCREENMIX(), SCREENATTR(), and the CA-Clipper commands SAVE/RESTORE SCREEN.

TODO: add documentation

Status

Started

Platforms

All

Files

Source is color.prg, library is libct.

ENHANCED()

Select the "ENHANCED" color value for output

Syntax

ENHANCED () -> <cEmptyString>

Description

TODO: add documentation

Status

Started

Compliance

ENHANCED() is compatible with CT3's ENHANCED()

Platforms

All

Files

Source is color.prg, library is libct.

See Also:

STANDARD()
UNSELECTED()

STANDARD()

Select the "STANDARD" color value for output

Syntax

```
STANDARD () -> <cEmptyString>
```

Description

TODO: add documentation

Status

Started

Compliance

STANDARD() is compatible with CT3's STANDARD()

Platforms

All

Files

Source is color.prg, library is libct.

See Also:

ENHANCED()
UNSELECTED()

UNSELECTED()

Select the "UNSELECTED" color value for output

Syntax

```
UNSELECTED () -> <cEmptyString>
```

Description

TODO: add documentation

Status

Started

Compliance

UNSELECTED() is compatible with CT3's UNSELECTED()

Platforms

All

Files

Source is color.prg, library is libct.

See Also:

ENHANCED()
STANDARD()

SCREENMIX()

$SYNTAX$

Description

TODO: add documentation

Status

Started

Platforms

All

Files

Source is screen2.prg, library is libct.