

ARx_Func4.ag

COLLABORATORS

	<i>TITLE :</i> ARx_Func4.ag		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 3, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	ARx_Func4.ag	1
1.1	"	1
1.2	ARexxGuide Functions reference (10 of 12) MESSAGE PORT	1
1.3	ARexxGuide Functions reference Message Ports (1 of 7) CLOSEPORT	2
1.4	ARexxGuide Functions reference Message Ports (2 of 7) GETARG	2
1.5	ARexxGuide Functions reference Message Ports (3 of 7) GETPKT	3
1.6	ARexxGuide Functions reference Message Ports (4 of 7) OPENPORT	3
1.7	ARexxGuide Functions reference Message Ports (5 of 7) REPLY	4
1.8	ARexxGuide Functions reference Message Ports (6 of 7) TYPEPKT	4
1.9	ARexxGuide Functions reference Message Ports (7 of 7) WAITPKT	5
1.10	ARexxGuide Functions reference (11 of 12) LOW-LEVEL	5
1.11	ARexxGuide Functions reference Low-level (1 of 13) ALLOCMEM	7
1.12	ARexxGuide Functions reference Low-level (2 of 13) BADDR	8
1.13	ARexxGuide Functions reference Low-level (3 of 13) EXPORT	8
1.14	ARexxGuide Functions reference Low-level (4 of 13) FORBID	9
1.15	ARexxGuide Functions reference Low-level (5 of 13) FREEMEM	10
1.16	ARexxGuide Functions reference Low-level (6 of 13) FREESPACE	10
1.17	ARexxGuide Functions reference Low-level (7 of 13) GETSPACE	11
1.18	ARexxGuide Functions reference Low-level (8 of 13) IMPORT	11
1.19	ARexxGuide Functions reference Low-level (9 of 13) NEXT	12
1.20	ARexxGuide Functions reference Low-level (10 of 13) NULL	13
1.21	ARexxGuide Functions reference Low-level (11 of 13) OFFSET	13
1.22	ARexxGuide Functions reference Low-level (12 of 13) PERMIT	14
1.23	ARexxGuide Functions reference Low-level (13 of 13) STORAGE	14
1.24	ARexxGuide Functions reference (12 of 12) BIT MANIPULATION	15
1.25	ARexxGuide Functions reference Bit-wise (1 of 8) BITAND	16
1.26	ARexxGuide Functions reference Bit-wise (2 of 8) BITCHG	16
1.27	ARexxGuide Functions reference Bit-wise (3 of 8) BITCLR	17
1.28	ARexxGuide Functions reference Bit-wise (4 of 8) BITCOMP	17
1.29	ARexxGuide Functions reference Bit-wise (5 of 8) BITOR	17
1.30	ARexxGuide Functions reference Bit-wise (6 of 8) BITSET	18
1.31	ARexxGuide Functions reference Bit-wise (7 of 8) BITTST	18
1.32	ARexxGuide Functions reference Bit-wise (8 of 8) BITXOR	19

Chapter 1

ARx_Func4.ag

1.1 "

AN AMIGAGUIDE® TO ARexx
by Robin Evans

Second edition (v2.0)

Note: This is a subsidiary file to ARexxGuide.guide. We recommend using that file as the entry point to this and other parts of the full guide.

Copyright © 1993,1994 Robin Evans. All rights reserved.

1.2 ARexxGuide | Functions reference (10 of 12) | MESSAGE PORT

CLOSEPORT
(<name>)

GETARG
(<packet>, [<number>])

GETPKT
(<name>)

OPENPORT
(<name>)

REPLY
(<packet>, <rc>)

TYPEPKT
(<name>)

WAITPKT
(<name>)

Also see ARexx control functions

Message ports are the primary means of communication among the many tasks and processes running on an Amiga. The ARexx resident process uses message ports extensively both for its own communication with the OS and to allow scripts to send commands to other environments.

These functions let an ARexx script set up and maintain its own message ports. The functions do not provide the level of control possible from a lower-level language like C, but they do allow for useful and powerful interaction among different scripts.

Next: Low-level func. | Prev: ARexx control func. | Contents: Function ref.

1.3 ARexxGuide | Functions reference | Message Ports (1 of 7) | CLOSEPORT

a rexxsupport.library ↔
function

```
rv = CLOSEPORT(<name>)
    rv is a Boolean value
```

Closes the port opened as <name>. The port must have been opened within the current ARexx program through a call to

```
OPENPORT()
```

.

Technique note: Use message ports in a script

Compatibility issues:

All support functions are system specific.

Next: GETARG() | Prev: VALUE() | Contents: Port mgt. func.

1.4 ARexxGuide | Functions reference | Message Ports (2 of 7) | GETARG

a rexxsupport.library ↔
function

```
rv = GETARG(<packet>, [<number>])
    rv is a string
```

Extracts a command, function name, or argument string from a message packet. The <packet> argument must be a valid address string obtained from a prior call to

```
GETPKT()
```

. If <number> is specified, then only the argument in that position is extracted. <number> must be less than or equal to the argument count for the packet.

Technique note: Use message ports in a script

Compatibility issues:

All support functions are system specific.

Next: GETPKT() | Prev: CLOSEPORT() | Contents: Port mgt. func.

1.5 ARexxGuide | Functions reference | Message Ports (3 of 7) | GETPKT

a rexxsupport.library ↔
function

```
rv = GETPKT(<name>)
    rv is an address string
```

Returns the address string of a message packet queued at the <name>d port. The message port must have been opened within the current ARexx program by a call to

```
OPENPORT()
```

. If no messages are available, the returned value will be '0000 0000'x (which is the same as

```
NULL()
```

```
).
```

Technique note: Use message ports in a script

Also see

```
WAITPKT
```

Compatibility issues:

All support functions are system specific.

Next: OPENPORT() | Prev: GETARG() | Contents: Port mgt. func.

1.6 ARexxGuide | Functions reference | Message Ports (4 of 7) | OPENPORT

a rexxsupport.library ↔
function

```
rv = OPENPORT(<name>)
    rv is an address string
```

Creates a public message port with the specified (and case-sensitive) <name>. A null address ('0000 0000'x) is returned if the port could not be initialized.

Technique note: Use message ports in a script

Also see

```
CLOSEPORT
```

```
WAITPKT
```

Compatibility issues:

All support functions are system specific.

Next: REPLY() | Prev: GETPKT() | Contents: Port mgt. func.

1.7 ARexxGuide | Functions reference | Message Ports (5 of 7) | REPLY

a rexxsupport.library ↔
function

```
rv = REPLY(<packet>, <rc>)
    rv is insignificant
```

A message packet with the primary result field set to the value given by <rc> is sent to <packet>, which must be a valid 4-byte address (usually obtained by a prior call to
OPENPORT()
.

Technique note: Use message ports in a script

Also see

GETPKT

Compatibility issues:

All support functions are system specific.

Next: TYPEPKT() | Prev: OPENPORT() | Contents: Port mgt. func.

1.8 ARexxGuide | Functions reference | Message Ports (6 of 7) | TYPEPKT

a rexxsupport.library ↔
function

```
rv = TYPEPKT(<address>, [<mode>])
    rv is a string
        or a number
        or a Boolean value
```

Although it is rarely needed in message ports handled by the current version of ARexx, this function returns information about a message packet received at the <address>. The <address> argument must be a valid address string usually obtained from
WAITPKT()
.

When the <mode> option is omitted, the function returns a packed 4-byte value, which can be unpacked to obtain information that is (with one exception) also available by specifying a mode argument. The meaning of each byte is explained below:

The mode arguments (which may be specified with only the first letter) are:

Mode	Information provided
-----	-----
Arguments	Returns the number of arguments. This information is contained in byte 0 of the unpacked return string.

Command Returns TRUE (1) if the packet was called as a command.
 This information is contained in byte 3 of the return string, which has a value of '01'x for commands.

Function Returns TRUE (1) if the packet was called as a function.
 This information is contained in byte 3 of the return , string, which has a value of '02'x for functions.

Byte 2 of the packed return string specifies the modifier flags that were set when the packet was called. The

REPLY()
 function automatically

handles any of the modifiers set by the calling command or function.

Because a script written with the current version of ARexx cannot serve as a reliable function host, calls to a port opened with an ARexx script should be sent as commands (which have a single argument string by default). That makes this function somewhat superfluous, but since it echoes an interface function that is genuinely useful in those other languages, it can be useful in prototyping an ARexx interface that will be transferred to a lower-level language

Compatibility issues:

All support functions are system specific.

Next: WAITPKT() | Prev: REPLY() | Contents: Port mgt. func.

1.9 ARexxGuide | Functions reference | Message Ports (7 of 7) | WAITPKT

a rexxsupport.library ↔
 function

rv = WAITPKT(<name>)
 rv is a Boolean value

Waits for a message to be received at the <name>d port which must have been opened with a prior call to

OPENPORT()
 . The function
 GETPKT()
 is used to retrieve the packet.

Technique note: Use message ports in a script

Compatibility issues:

All support functions are system specific.

Next: Port mgt. func. | Prev: TYPEPKT() | Contents: Port mgt. func.

1.10 ARexxGuide | Functions reference (11 of 12) | LOW-LEVEL


```
ALLOCMEM
(<length>, [<attribute>])

BADDR
(<BCPL address string>)

EXPORT
(<address>, [<string>], [<length>], [<padchar>])

FORBID
()

FREEMEM
(<address>, <length>)

FREESPACE
([<address>, <length>])

GETSPACE
(<length>)

IMPORT
(<address>, [<length>])

NEXT
(<address>, [<offset>])

NULL
()

OFFSET
(<address>, <displacement>)

PERMIT
()

STORAGE
([<address>], [<string>], [<length>], [<padchar>])
```

Related function:

SHOWLIST

Most ARexx scripts will never need these functions since the ARexx resident process takes care of things like the memory allocations needed to store variable references.

The functions in this list will be familiar to those who use assembler or C languages to program the machine since they closely parallel the similarly-named Amiga system functions that are used extensively in those environments. That's probably one reason they are included in the support library. They provide a useful tool for prototyping a program -- a way to write an early version of a program in ARexx, an interpreted language that allows quick and simple changes and has powerful debugging tools. These support functions allow a programmer to test program logic and effectiveness in ARexx before committing the code to a compiled language.

With care, they may also be used in any ARexx script that needs special access to aspects of the OS not normally available in ARexx. Note, though, that these are the most dangerous functions included in the ARexx package since many of them circumvent the checks and balances usually provided by the ARexx resident process.

The address string

~~~~~

The <address> argument used by many of these functions must be a 4-byte address string. This is a four-character string and not a number as might be used in other languages. Use the `c2x()` or `c2d()` functions to translate the <address> into more readable form. (See note at

`OFFSET()`  
about computing new addresses with the character translation ←  
functions.)

The Amiga ROM Kernal Manuals explain in detail the system functions called by these ARexx functions. The Sullivan & Zamara book, *Using ARexx on the Amiga* is recommended reading for those who want more information about how these functions can be used in ARexx scripts.

Next: Bit-wise func. | Prev: Port mgt. func. | Contents: Function ref.

## 1.11 ARexxGuide | Functions reference | Low-level (1 of 13) | ALLOCMEM

a rexxsupport.library ←  
function

```
rv = ALLOCMEM(<length>, [<attribute>])
  rv is an address string
```

Allocates a block of memory of the specified <length> from the system free-memory pool.

<attribute> may be any of the standard flags used with the Exec AllocMem function, but must be supplied as a four-byte string. The default is public (`MEMF_PUBLIC`).

Example:

```
addr = allocmem(32);
call
    freemem(addr,32)
;
```

This support function calls the OS AllocMem() function. Care should be taken in using it since ARexx performs no special checks and will not automatically deallocate the memory block when the program exits.

Also see

`GETSPACE`

Compatibility issues:

All support functions are system specific.

Next: `BADDR()` | Prev: Low-level func. | Contents: Low-level func.

## 1.12 ARexxGuide | Functions reference | Low-level (2 of 13) | BADDR

a rexxsupport.library function

```
rv = BADDR(<BCPL address string>)
    rv is an address_string
```

Converts a BPTR to an CPTR address.

Compatibility issues:  
All support functions are system specific.

Next: EXPORT() | Prev: ALLOCMEM() | Contents: Low-level func.

## 1.13 ARexxGuide | Functions reference | Low-level (3 of 13) | EXPORT

a rexxsupport.library ↔  
function

```
rv = EXPORT(<address>, [<string>], [<length>], [<padchar>])
    rv is a number
```

Copies data from the optional <string> into the area starting at <address>. Sufficient memory should have been previously allocated with a call to

```
ALLOCMEM()
or
GETSPACE()
.
```

If <string> is shorter than <length>, then the <padchar> (which defaults to a null) will be used to fill out the space.

The value returned is the number of bytes written to memory. A pointer to the new address can be obtained by adding the value returned to the current address using

```
OFFSET()
.
```

Example:

```
/* copies a file from disk and stores it in memory          **
** A file, 'Input', must have been opened previously, and an **
** address [CurAddr] obtained through                      **
    getspace()
    .
    */
do until eof('Input')
    /* readch() reads the file, either the whole thing, or  **
    ** the first 64k bytes                                   **
    ** export() stores value returned by readch() in memory. **
    ** offset() computes the next address based on the return **
    ** from export().                                       **
    CurAddr = offset(CurAddr, export(CurAddr, readch('Input', 65535)))
```

end

Also see

STORAGE

IMPORT

Compatibility issues:

All support functions are system specific.

Next: FORBID() | Prev: BADDR() | Contents: Low-level func.

## 1.14 ARexxGuide | Functions reference | Low-level (4 of 13) | FORBID

a rexxsupport.library ↔  
function

```
rv = FORBID()
  rv is a number
```

Task switching can be controlled by calls to FORBID() and PERMIT(). The

return value is the current nesting count which is -1 when task-switching is enabled. Since ARexx scripts run as separate tasks, no harm is done if the program ends with task switching forbidden, but it is good practice to enable multitasking as quickly as possible.

This function should be used whenever items are read from the various system lists since, if multitasking is enabled, another task might cause a change in the list while it is being read by the ARexx script. The Amiga ROM Kernal Manuals warn, "To access these lists without forbidding jeopardizes the integrity of the entire system." The warning applies to any language, so this is not something unique to ARexx.

The interpreter will handle calls to forbid() and permit() when the SHOWLIST() function is used.

All forms of I/O should be avoided while FORBID() is in effect, since any kind of I/O (including instructions like SAY and PULL, or functions like

WAITPKT()

and OPEN() ) will cause the system to wait for

I/O completion and to disable the forbidden state while it waits.

See example at

IMPORT()

Technique note: Determine library version number

Compatibility issues:

All support functions are system specific.

Next: FREEMEM() | Prev: EXPORT() | Contents: Low-level func.

## 1.15 ARexxGuide | Functions reference | Low-level (5 of 13) | FREEMEM

a rexxsupport.library ↔  
function

```
rv = FREEMEM(<address>,<length>)
  rv is a Boolean value
```

Releases the block of memory of <length> (an integer) size at <address> to the system freelist. <address> must be a valid 4-byte address, usually obtained by a prior call to  
ALLOCMEM()  
.

Example:

```
addr = allocmem(32);
call freemem(addr,32);
```

Also see

FREESPACE

Compatibility issues:

All support functions are system specific.

Next: FREESPACE() | Prev: FORBID() | Contents: Low-level func.

## 1.16 ARexxGuide | Functions reference | Low-level (6 of 13) | FREESPACE

a rexxsupport.library ↔  
function

```
rv = FREESPACE([<address>, <length>])
  rv is a Boolean value
  or a number
```

Releases to the internal pool maintained by the interpreter the block of memory of <length> (an integer) size at <address> (which should have been obtained through a previous call to

GETSPACE()

) . If called without

arguments, the function returns the amount of memory available in the interpreter's internal pool.

The interpreter will release the memory when a script ends even if this function is not called.

Also see

FREEMEM

Compatibility issues:

All support functions are system specific.

Next: GETSPACE() | Prev: FREEMEM() | Contents: Low-level func.

## 1.17 ARexxGuide | Functions reference | Low-level (7 of 13) | GETSPACE

a rexxsupport.library ↔  
function

```
rv = GETSPACE(<length>)
  rv is an address string
```

Allocates a block of memory of <length> size (a decimal number) from the interpreter's internal pool.

The memory is automatically returned to the system when the ARexx script that calls this function terminates.

Example:

```
/**/
MemWant = 19764
  /* Is there enough memory for the allocation? */
if storage() > MemWant then
  StoreAddress = getspace(MemWant)
else
  say 'Not enough memory.'
```

Also see

FREESPACE

ALLOCMEM

STORAGE

Compatibility issues:

All support functions are system specific.

Next: IMPORT() | Prev: FREESPACE() | Contents: Low-level func.

## 1.18 ARexxGuide | Functions reference | Low-level (8 of 13) | IMPORT

a rexxsupport.library ↔  
function

```
rv = IMPORT(<address>, [<length>])
  rv is a string
```

A string of <length> (an integer) bytes is returned. It is copied from memory starting at the specified <address> (which must be specified as a 4-byte address string. If <length> is not specified, values will be copied until a null byte is encountered.

Example:

```
/* Imports name and size of default font */
gfxbase=showlist(1, 'graphics.library',,a)
call forbid
FntAddr = next(gfxbase,154)
DefFont = IMPORT(next(FntAddr, 10))
FSize = c2d(IMPORT(offset(FntAddr, 20),2))
call permit
```

Also see

```
EXPORT
    SHOWLIST

NEXT

OFFSET

FORBID

PERMIT
```

Technique note: Determine library version number

Compatibility issues:

All support functions are system specific.

Next: NEXT() | Prev: GETSPACE() | Contents: Low-level func.

## 1.19 ARexxGuide | Functions reference | Low-level (9 of 13) | NEXT

a rexxsupport.library ↔  
function

```
rv = NEXT(<address>,[<offset>])
    rv is an address string
```

Returns the 4-byte address string at <address> plus <offset>. The function combines features of

```
import()
and
offset()
```

. Like import(), it reads a value from memory, but is designed for the specific task obtaining an address. Like offset(), it will, when given a decimal offset, calculate a new address in the proper format.

A linked-list maintained by the operating system can be followed by using the following format:

```
NextNode = NEXT(<node-address>)
PrevNode = NEXT(<node-address>, 4).
```

See example at

```
IMPORT()
Also see
OFFSET
```

The base address of most system resources can be obtained with the SHOWLIST() function, using its fourth 'Address' argument.

Compatibility issues:

All support functions are system specific.

Next: NULL() | Prev: IMPORT() | Contents: Low-level func.

## 1.20 ARexxGuide | Functions reference | Low-level (10 of 13) | NULL

a rexxsupport.library ↔  
function

```
rv = NULL()
    rv is an address string
```

The result is a null pointer as a 4-byte string ('0000 0000'x).

Also see

OFFSET

Technique note: Use message ports in a script

Compatibility issues:

All support functions are system specific.

Next: OFFSET() | Prev: NEXT() | Contents: Low-level func.

## 1.21 ARexxGuide | Functions reference | Low-level (11 of 13) | OFFSET

a rexxsupport.library ↔  
function

```
rv = OFFSET(<address>,<displacement>)
    rv is an address string
```

Computes a new address as the signed offset from a base address.

<address> must be a valid 4-byte address string. <displacement> must be an integer in decimal (not hexadecimal) form.

This function will compute the address of a field in a data structure without requiring calls to C2D() and D2C() , and does so in a safer way since -- unlike the return value of the translation functions -- the value returned by offset() will always be a 4-byte address string.

Example:

```
say c2x(offset('0000 0000'x,4)) >>> 00000676
```

See example at

IMPORT()

Also see

NEXT

NULL

C2X

Technique note: Determine library version number

Compatibility issues:



All support functions are system specific.

Next: PERMIT() | Prev: NULL() | Contents: Low-level func.

## 1.22 ARexxGuide | Functions reference | Low-level (12 of 13) | PERMIT

a rexxsupport.library ↔  
function

```
rv = PERMIT()
    rv is a number
```

Lowers by one the nesting count of  
FORBID()  
and returns the current  
nesting count.

Each call to FORBID() will raise the nesting count by 1 from its base count of -1. When the count reaches -1 after successive calls to FORBID() and PERMIT(), multitasking will be possible once again.

If a task (a script) ends in a forbidden state, no harm is done since the state is cleared automatically when the task ends, but it is good practice to call PERMIT() as quickly as possible after entering a forbidden state.

See example at  
IMPORT()

Technique note: Determine library version number

Compatibility issues:  
All support functions are system specific.

Next: STORAGE() | Prev: OFFSET() | Contents: Low-level func.

## 1.23 ARexxGuide | Functions reference | Low-level (13 of 13) | STORAGE

a rexxsupport.library ↔  
function

```
rv = STORAGE([<address>], [<string>], [<length>], [<padchar>])
    rv is a number
```

If all arguments are omitted, the function returns the amount of free memory in the system.

If <address> is given (as a valid 4-byte address string) then data from <string> will be copied to that address for <length> (an integer) bytes. If <string> is shorter than <length>, then the space will be filled with <padchar>.

The default pad character is a null.

Examples:

```
say storage()          >>> 7121608
```

Also see

EXPORT

IMPORT

Compatibility issues:

All support functions are system specific.

Next: Low-level func. | Prev: PERMIT() | Contents: Low-level func.

## 1.24 ARexxGuide | Functions reference (12 of 12) | BIT MANIPULATION

```
BITAND
(<string1>,<string2>,[<padchar>])
```

```
BITCHG
(<string>,<bit>)
```

```
BITCLR
(<string>,<bit>)
```

```
BITCOMP
(<string>,<string2>,[<padchar>])
```

```
BITOR
(<string1>,[<string2>],[<padchar>])
```

```
BITSET
(<string>,<bit>)
```

```
BITTST
(<string>,<bit>)
```

```
BITXOR
(<string1>,<string2>,[<padchar>])
```

Also see Comparison functions  
Number manipulation functions

The primary argument to each of these functions, and the value returned by most of them is a character or character string. The functions work at the low bit-level so familiar to those who program in assembly language. The binary representation of the character 'a', for example, is '01100001' in the ASCII character set which can be expressed as '01100001'b or as c2b('a'). The function BITSET() can change just one bit in that 'field'. BITSET('a', 1) will return 'c' -- the character with the binary representation of '01100011'.

A string 0's and 1's is not a proper argument to any of these functions since the 0 would be interpreted as ASCII character 48 (decimal) or 00110000 (binary). Using a binary string, on the other hand, will cause

the 0's and 1's to be translated to the character representation expected by the functions.

Compatibility issues :

Some examples in the following nodes use alphabetic characters to show how a function works. The results shown here are valid only for the ASCII character set. Since REXX was developed and is still used largely on systems that use a different character set, such examples are frowned upon in the REXX standard.

Next: [Function ref.](#) | [Prev: Low-level func.](#) | [Contents: Function ref.](#)

## 1.25 ARexxGuide | Functions reference | Bit-wise (1 of 8) | BITAND

```
rv = BITAND(<string1>,<string2>, [<padchar>])
rv is a string
```

The result is equal to the length of longer of the two supplied strings which are logically AND'ed together bit by bit. If a pad character is supplied, then the shorter string is filled out with that character until it is the same length as the other string.

The default <padchar> is the null character.

Examples:

```
say bitand('A', 'J') >>> @
say bitand('01000001'b, '01001010'b) >>> @
say c2b(bitand('01000001'b, '01001010'b)) >>> 01000000
```

Next: [BITCHG\(\)](#) | [Prev: Bit-wise func.](#) | [Contents: Bit-wise func.](#)

## 1.26 ARexxGuide | Functions reference | Bit-wise (2 of 8) | BITCHG

```
rv = BITCHG(<string>, <bit>)
rv is a string
```

The state of the specified <bit> in <string> is changed. Bit 0 is the low-order bit of the rightmost byte of the string.

Examples:

```
/**/
say bitchg('a', 5) >>> A
say bitchg('A', 5) >>> a
say c2b(bitchg('01101100'b, 3)) >>> 01100100
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: [BITCLR\(\)](#) | [Prev: BITAND\(\)](#) | [Contents: Bit-wise func.](#)

## 1.27 ARexxGuide | Functions reference | Bit-wise (3 of 8) | BITCLR

```
rv = BITCLR(<string>, <bit>)
rv is a string
```

The specified <bit> in <string> is cleared (set to 0). Bit 0 is the low-order bit of the rightmost byte of the string.

Examples:

```
/**/
say bitclr('a', 5)           >>> A
say bitclr('A', 5)           >>> A
say c2b(bitchg('01101100'b, 3)) >>> 01100100
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: BITCOMP() | Prev: BITCHG() | Contents: Bit-wise func.

## 1.28 ARexxGuide | Functions reference | Bit-wise (4 of 8) | BITCOMP

```
rv = BITCOMP(<string>, <string2>, [<padchar>])
rv is a number
```

The result indicates the first position of the bit at which the two supplied strings differ or -1 if they are the same. The shorter string is padded with <padchar> before the comparison.

The default pad character is a null.

Examples:

```
say bitcomp('0011'b, '0111'b)           >>> 2
say bitcomp('c', 'C')                   >>> 5
say bitcomp('b', 'B')                   >>> 5
say bitcomp('0a'x, '1a'x)               >>> 4
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: BITOR() | Prev: BITCLR() | Contents: Bit-wise func.

## 1.29 ARexxGuide | Functions reference | Bit-wise (5 of 8) | BITOR

```
rv = BITOR(<string1>, [<string2>], [<padchar>])
rv is a string
```

The result is equal to the length of longer of the two supplied strings which are logically (inclusively) OR'ed together bit by bit. If a pad

character is supplied, then the shorter string is filled out with that character until it is the same length as the other string.

The default <padchar> is the null character.

Example:

```
say bitor('A', 'J')           >>> K
say bitor('01000001'b, '01001010'b) >>> K
say bitor('Amiga FOREVER')    >>> amiga forever
say bitor('FRANÇOIS')        >>> françios
say bitor('THIS_THAT')       >>> thisthat
```

It is unsafe to use bitor() for character translation since characters like '[' and '\_' (ASCII 91 to 96) that come between 'Z' and 'a' in the ASCII set are treated improperly. The TRANSLATE() function provides a safe way to implement a user-defined lower() function.

Next: BITSET() | Prev: BITCOMP() | Contents: Bit-wise func.

### 1.30 ARexxGuide | Functions reference | Bit-wise (6 of 8) | BITSET

```
rv = BITSET(<string>, <bit>)
rv is a string
```

The specified <bit> in <string> is set to 1.

Examples:

```
say bitset('A', 5)           >>> a
say bitset('00101'b, 3)      >>> 00101
say c2b(bitset('0000101'b, 3)) >>> 00001101
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: BITTST() | Prev: BITOR() | Contents: Bit-wise func.

### 1.31 ARexxGuide | Functions reference | Bit-wise (7 of 8) | BITTST

```
rv = BITTST(<string>, <bit>)
rv is a Boolean value
```

The result indicates the state of the specified <bit> in <string>.

Examples:

```
"say bittst('00001001'b, 3)    >>> 1 (True)
"say bittst('00001001'b, 1)    >>> 0 (False)
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations,

there is no assurance that it will be.

Next: BITXOR() | Prev: BITSET() | Contents: Bit-wise func.

## 1.32 ARexxGuide | Functions reference | Bit-wise (8 of 8) | BITXOR

```
rv = BITXOR(<string1>,<string2>,[<padchar>])  
rv is a string
```

The result is equal to the length of longer of the two supplied strings which are logically (exclusively) OR'ed together bit by bit. If a pad character is supplied, then the shorter string is filled out with that character until it is the same length as the other string.

The default <padchar> is the null character.

Examples:

```
say c2b(bitxor('00001101'b, '01000101'b)) >>> 01001000  
say bitxor('00001101'b, '01000101'b) >>> H  
say c2b(bitxor('A', 'J')) >>> 00001011
```

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: Bit-wise func. | Prev: BITTST() | Contents: Bit-wise func.

---