ARx_Func4.ag

COLLABORATORS			
	TITLE : ARx_Func4.ag		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 8, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Func4.ag

1.1 main

```
AN AMIGAGUIDE® TO ARexx                 Edition: 1.0a
by Robin Evans
```

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, 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 and operating system. 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
CLOSEPORT(<name>)
      returns 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() .

    Ports example
```

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

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

```
                                        a rexxsupport.library function
GETARG(<packet>, [<number>])
      returns a string

Extracts a command, function name, or argument string from a message
packet. The <packet> argument must be a valid 4-byte address 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.

    Ports example
```

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

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

```
                                        a rexxsupport.library function
GETPKT(<name>)
      returns a 4-byte address string

Returns the 4-byte address 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() ).

    Ports example

   Also see @{ " WAITPKT       " link WAITPKT()}
```

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

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

```
                                    a rexxsupport.library function
OPENPORT(<name>)
      returns a 4-byte 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.

    Ports example

   Also see @{ " CLOSEPORT     " link CLOSEPORT()}
             @{ " WAITPKT      " link WAITPKT()}

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

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

```
                                    a rexxsupport.library function
REPLY(<packet>, <rc>)
      return value 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() .

    Ports example

   Also see @{ " GETPKT        " link GETPKT()}

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

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

```
                                    a rexxsupport.library function
TYPEPKT(<name>, [<mode>])
      returns 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
sent to the port opened as <name>.

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 modifyer flags that
were set when the packet was called. The  REPLY()  fuction 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. It can,
nonetheless, be useful in  prototyping  an ARexx interface that will be
transferred to a lower-level language since it echos an interface function
that is genuinely useful in those other languages.

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

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

```
                                        a rexxsupport.library function
WAITPKT(<name>)
      returns 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() must be used to actually retrieve the packet.

    Ports example

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 Amiga ROM Kernal Manuals explain in detail the usage of 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
ALLOCMEM(<length>, [<attribute>])
      returns a 4-byte address string

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

   Example:
         addr = allocmem(32);
         call  freemem(addr,32) ;

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

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

```
BADDR(<BCPL address string>)
      returns a 4-byte address string
```

Converts a BPTR to an CPTR address.

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

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

a rexxsupport.library function

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

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.

```
   Also see @{ " STORAGE       " link STORAGE()}
            @{ " IMPORT        " link IMPORT()}
```

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

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

a rexxsupport.library function

```
FORBID()
      returns a number
```

Task switching can be controlled by calls to FORBID() and PERMIT() . The return value is the current nesting count (or -1 if task switching is enabled). Since ARexx programs run as separate tasks, no harm is done if the program ends with task switching forbidden.

```
   See example at  IMPORT()
```

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

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

a rexxsupport.library function

```
FREEMEM(<address>,<length>)
      returns a Boolean value
```

Releases the block of memory of <length> size at <address> from 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);

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

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

```
                                       a rexxsupport.library function
FREESPACE([<address>, <length>])
      returns a Boolean value
           or a number

Releases the block of memory of <length> size at <address> (which should
have been obtained through a previous call to  GETSPACE()  to the
internal pool maintained by the  interpreter . Calling the function without
arguments will return the amount of memory available in the interpreter's
internal pool.

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

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

```
                                       a rexxsupport.library function
GETSPACE(<length>)
      returns a 4-byte address string

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

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

   Also see @{ " FREESPACE     " link FREESPACE()}

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

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

```
                                       a rexxsupport.library function
IMPORT(<address>, [<length>])
      returns a string

The result is created by copying values from the specified <address> for
<length> bytes. 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(l, '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        " link EXPORT()}
            @{ " SHOWLIST      " link ARx_Func2.ag/SHOWLIST()}
            @{ " NEXT          " link NEXT()}
            @{ " OFFSET        " link OFFSET()}
            @{ " FORBID        " link FORBID()}
            @{ " PERMIT        " link PERMIT()}

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

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

```
                                    a rexxsupport.library function
NEXT(<address>,[<offset>])

Returns the 4-byte value at <address> (plus <offset>). It can be used to
follow a EXEC list forwards as NEXT(address) or backwards as
NEXT(address,4).

      See example at  IMPORT()

   Also see @{ " OFFSET        " link OFFSET()}

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

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

```
                                    a rexxsupport.library function
NULL()
      returns a 4-byte address string

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

   Also see @{ " OFFSET        " link OFFSET()}

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

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

```
                                    a rexxsupport.library function
OFFSET(<address>,<displacement>)
      returns a 4-byte address string

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

address argument must be a 4-byte string, and the displacement argument
is a decimal integer.

This function will compute the address of a field in a data structure
without requiring calls to C2D() and D2C().

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

      See example at IMPORT()

   Also see @{ " NEXT   " link NEXT()}
            @{ " NULL   " link NULL()}

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

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

                                    a rexxsupport.library function
PERMIT()
      returns a number

Task switching can be controlled by calls to FORBID() and PERMIT(). The
return value is the current nesting count (or -1 if task switching is
enabled). Since ARexx programs run as separate tasks, no harm is done if
the program ends with task switching forbidden.

      See example at IMPORT()

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

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

                                    a rexxsupport.library function
STORAGE([<address>], [<string>], [<length>],[<padchar>])
      returns 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> 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   " link EXPORT()}
            @{ " IMPORT   " link IMPORT()}

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' 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'. Note that a string 0's and 1's is not a proper argument to any of these functions. They will treat the 0 as ASCII character 48 (decimal) or 00110000 (binary). The use of 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.

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

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

```
BITAND(<string1>,<string2>, [<padchar>])
      returns a character 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.

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

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

```
BITCHG(<string>, <bit>)
      returns a character string

The state of the specified <bit> in <string> is changed. Bit 0 is 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

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

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

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

The specified <bit> in <string> is cleared (set to 0). Bit 0 is 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

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

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

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

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

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

```
BITOR(<string1>,[<string2>],[<padchar>])
      returns a character 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('Amiga FOREVER')             >>> amiga forever

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

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

```
BITSET(<string>, <bit>)
      returns a character string

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

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

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

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

```
BITTST(<string>, <bit>)
      returns a Boolean value

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

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

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

```
BITXOR(<string1>,<string2>,[<padchar>])
      returns a character 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.

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