AREXXLIB

Conversion program

COLLABORATORS			
	TITLE : AREXXLIB		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Conversion program	February 2, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AREXXLIB

1.1 Overview of AREXXLIB

```
                                   Overview


                             An Acid Software Library

                            Converted to AmigaGuide by

                               Red When Excited Ltd

                      Used with the permission of Acid Software
```

1.2 AREXXLIB

```
Statement: CreateRexxMsg
--------------------------------------------------------------------------------
Modes  :
Syntax : CreateRexxMsg MsgPtr=CreateRexxMsg(ReplyPort,Exten,Host)

MODES:AMIGA

DESCRIPTION:

CreateRexxMsg() allocates a special Message structure used to communicate
with Arexx. If all is successful it returns the LONGWORD address of this
rexxmsg structure.

The arguments are ReplyPort which is the long address returned by
CreateMsgPort(). This is the Port that ARexx will reply to after it has
finished with the message.

EXTEN which is the exten name used by any ARexx script you are wishing to
run. i.e. if you are attempting to run the ARexx script test.rexx you would
use an EXTEN of "rexx".
```

HOST is the name string of the HOST port. Your program is usually the HOST and so this equates to the name you gave your port in CreateMsgPort(). REMEMBER IT IS CASE SENSITIVE.

As we are allocating resources error checking is important and can be achieved with the following code:

```
msg.l=CreateRexxMsg(Port,"rexx","HostName")
IF msg=0 THEN Error_Routine{}
```

1.3 AREXXLIB

```
Statement: DeleteRexxMsg
--------------------------------------------------------------------------------
Modes  :
Syntax : DeleteRexxMsg rexxmsg.l

MODES:AMIGA

DESCRIPTION:
```

DeleteRexxMsg simply deletes a RexxMsg Structure previously allocated by CreateRexxMsg(). It takes a single argument which is the long address of a RexxMsg structure such as returned by CreateRexxMsg().

```
msg.l=CreateRexxMsg(Port,"rexx","HostName")
IF msg=0 THEN Error_Routine{}
DeleteRexxMsg msg
```

Again if you neglect to delete the RexxMsg structure Blitz2 will do this for you on exit of the program.

1.4 AREXXLIB

```
Statement: FillRexxMsg
--------------------------------------------------------------------------------
Modes  :
Syntax : FillRexxMsg rexxmsg,&FillStruct

MODES:AMIGA

DESCRIPTION:
```

FillRexxMsg allows you to fill all 16 ARGSlots if necessary with either ArgStrings or numerical values depending on your requirement.

FillRexxMsg will only be used by those programmers wishing to do more advanced things with Arexx, including adding libraries to the ARexx library list, adding Hosts,Value Tokens etc. It is also needed to access Arexx using the #RXFUNC flag.

The arguments are a LONG Pointer to a rexxmsg.

The LONG address of a FillStruct NEWTYPE structure. This structure is defined in the Arexx.res and has the following form.

```
NEWTYPE.FillStruct
  Flags.w       ;Flag block
  Args0.l       ; argument block (ARG0-ARG15)
  Args1.l       ; argument block (ARG0-ARG15)
  Args2.l       ; argument block (ARG0-ARG15)
  Args3.l       ; argument block (ARG0-ARG15)
  Args4.l       ; argument block (ARG0-ARG15)
  Args5.l       ; argument block (ARG0-ARG15)
  Args6.l       ; argument block (ARG0-ARG15)
  Args7.l       ; argument block (ARG0-ARG15)
  Args8.l       ; argument block (ARG0-ARG15)
  Args9.l       ; argument block (ARG0-ARG15)
  Args10.l       ; argument block (ARG0-ARG15)
  Args11.l       ; argument block (ARG0-ARG15)
  Args12.l       ; argument block (ARG0-ARG15)
  Args13.l       ; argument block (ARG0-ARG15)
  Args14.l       ; argument block (ARG0-ARG15)
  Args15.l       ; argument block (ARG0-ARG15)
  EndMark.l      ;End of the FillStruct
End NEWTYPE
```

The Args?.l are the 16 slots that can possibly be filled ready for converting into the RexxMsg structure. The Flags.w is a WORD value representing the type of LONG word you are supplying for each ARGSLOT (Arg?.l).

Each bit in the Flags WORD is representative of a single Args?.l, where a set bit represents a numerical value to be passed and a clear bit represents a string argument to be converted into a ArgString before installing in the RexxMsg. The Flags Value is easiest to supply as a binary number to make the bits visible and
would look like this.

```
%0000000000000000 ;This represents that all Arguments are Strings.

%0110000000000000 ;This represent the second and third as being integers.
```

FillRexxMsg expects to find the address of any strings in the Args?.l slots so it is important to remember when filling a FillStruct that you must pass the string address and not the name of the string. This is acomplished using the '&' address of operand.

So to use FillRexxMsg we must do the following things in our program:

1. Allocate a FillStruct
2. Set the flags in the FillStruct\Flags.w
3. Fill the FillStruct with either integer values or the
   addresses of our string arguments.
4. Call FillRexxMsg with the LONG address of our rexxmsg and the
   LONG address of our FillStruct.

```
To accomplish this takes the following code:

;Allocate our FillStruct (called F)

DEFTYPE.FillStruct F

;assign some string arguments

T$="open":T1$="0123456789"

;Fill in our FillStruct with flags and (&) addresses of our strings

F\Flags= %0010000000000000,&T$,&T1$,4

;Third argument here is an integer (4).

Port.l=CreateMsgPort("host")
msg.l=CreateRexxMsg(Port,"vc","host")

FillRexxMsg msg,&F

 ;<-3 args see #RXFUNC

SendRexxCommand msg,"",#RXFUNC|#RXFF_RESULT|3
```

1.5 AREXXLIB

```
Statement: ClearRexxMsg
--------------------------------------------------------------------------------
Modes  :
Syntax : ClearRexxMsg rexxmsg

MODES:AMIGA

DESCRIPTION:

ClearRexxMsg is used to delete and clear an ArgString from one ormore of the
Argument slots in a RexxMsg Structure. This is most useful for the more
advanced programmer wishing to take advantage of the Arexx #RXFUNC
abilities.

The arguments are a LONGWORD address of a RexxMsg structure. ClearRexxMsg
will always work from slot number 1 forward to 16.

Port.l=CreateMsgPort("TestPort")
If Port = NULL Then End
msg.l=CreateRexxMsg(Port,"vc","TestPort")
If msg=NULL Then End
SendRexxCommand msg,"open",#RXCOMM|#RXFF_RESULT
wait:WHILE GetMsg_(Port) <> msg:Wend     ;Wait for reply to come
ClearRexxMsg msg     .      ;Delete the Command string we sent

NOTE: ClearRexxMsg() is called automatically by RexxEvent() so the need to
```

call this yourself is removed unless you have not sent the RexxMsg to Arexx.

1.6 AREXXLIB

```
Statement: CreateArgString
--------------------------------------------------------------------------------
Modes  :
Syntax : CreateArgString ArgString=CreateArgString(String)

MODES:AMIGA

DESCRIPTION:

CreateArgString() builds an ARexx compatible ArgString structure around the
provided string. All strings sent to, or received from Arexx are in the form
of ArgStrings. See the TYPE RexxARG.

If all is well the return will be a LONG address of the ArgString structure.
The pointer will actually point to the NULL terminated String with the
remainder of the structure available at negative offsets.

arg.l=CreateArgString("this is a string")
 IF arg=0 THEN Error_Routine{}:ENDIF
DeleteArgString arg

NOTE: An ArgString maybe used as a normal BB2 string variable by simple
conversion using PEEK$

i.e. msg$=PEEK$(arg) or perhaps NPRINT PEEK$(arg)

NOTE: Most of the BB2 Arexx Functions call this themselves and there will be
only limited need for you to access this function.
```

1.7 AREXXLIB

```
Statement: DeleteArgString
--------------------------------------------------------------------------------
Modes  :
Syntax : DeleteArgString argstring

MODES:AMIGA

DESCRIPTION:

DeleteArgString is designed to Delete ArgStrings allocated by either Blitz2
or ARexx in a system friendly way. It takes only one argument the LONGWORD
address of an ArgString as returned by CreateArgString().

arg.l=CreateArgString("this is a string")
 IF arg=0 THEN Error_Routine{}:ENDIF
```

DeleteArgString arg

NOTE: This function is also called automatically by most of the BB2 Arexx Functions that need it so you should only need to call this on rare occations.

1.8 AREXXLIB

Statement: SendRexxCommand

Modes :
Syntax : SendRexxCommand rexxmsg,CommandString,ActionCodes

```
SendRexxCommand rexxmsg,"commandstring",#RXCOMM|#RXFF_RESULT
```

MODES:AMIGA

DESCRIPTION:

SendRexxCommand is designed to fill and send a RexxMsg structure to ARexx inorder to get ARexx to do something on your behalf.

The arguments are as follows;

rexxmsg is the LONGWORD address of a RexxMsg structure as returned by CreateRexxMsg().

commandstring is the command string you wish to send to ARexx. This is a string as in "this is a string" and will vary depending on what you wish to do with ARexx. Normally this will be the name of an ARexx script file you wish to execute. ARexx will then look for the script by the name as well as the name with the exten added.(this is the exten you used when you created the RexxMsg structure using CreateRexxMsg()). This could also be a string file. That is a complete ARexx script in a single line.

ActionCodes are the flag values you use to tell ARexx what you want it to do with the commandstring you have supplied. The possible flags are as follows;

COMMAND (ACTION) CODES

The command codes that are currently implemented in the resident process are described below. Commands are listed by their mnemonic codes,followed by the valid modifier flags. The final code value is always the logical OR of the code value and all of the modifier flags selected. The command code is installed in the rm_Action field of the message packet.

USAGE: RXADDCON

This code specifies an entry to be added to the Clip List. Parameter slot ARGO points to the name string,slot ARG1 points to the value string,and slot ARG2 contains the length of the value string.

The name and value arguments do not need to be argstrings,but can be just pointers to storage areas. The name should be a null-terminated string,but the value can contain arbitrary data including nulls.

USAGE: RXADDFH

This action code specifies a function host to be added to the Library List. Parameter slot ARGO points to the (null-terminated) host name string,and slot ARG1 holds the search priority for the node. The search priority should be an integer between 100 and -100 inclusive;the remaining priority ranges are reserved for future extensions. If a node already exists with the same name,the packet is returned with a warning level error code.

Note that no test is made at this time as to whether the host port exists.

USAGE:RXADDLIB

This code specifies an entry to be added to the Library List. Parameter slot ARGO points to a null-terminated name string referring either to a function library or a function host. Slot ARG1 is the priority for the node and should be an integer between 100 and -100 inclusive;the remaining priority ranges are reserved for future extensions. Slot ARG2 contains the entry Point offset and slot ARG3 is the library version number. If a node already exists with the same name,the packet is returned with a warning level error code. Otherwise,a new entry is added and the library or host becomes available to ARexx programs. Note that no test is made at this time as to whether the library exists and can be opened.

USAGE:RXCOMM [RXFF_TOKEN] [RXFF_STRING] [RXFF_RESULT] [RXFF_NOIO]

Specifies a command-mode invocation of an ARexx program. Parameter slot ARGO must contain an argstring Pointer to the command string. The RXFB_TOKEN flag specifies that the command line is to be tokenized before being passed to the invoked program. The RXFB_STRING flag bit indicates that the command string is a "string file." Command invocations do not normally return result strings,but the RXFB_RESULT flag can be set if the caller is prepared to handle the cleanup associated with a returned string. The RXFB_NOIO modifier suppresses the inheritance of the host’s input and output streams.

USAGE:RXFUNC [RXFF_RESULT] [RXFF_STRING] [RXFF_NOIO] argcount

This command code specifies a function invoction. Parameter slot ARGO contains a pointer to the function name string,and slots ARG1 through ARG15 point to the argument strings,all of which must be passed as argstrings. The lower byte of the command code is the argument count;this count excludes the function name string itself. Function calls normally set the RXFB_RESULT flag,but this is not mandatory. The RXFB_STRING modifier indicates that the function name string is actually a "string file". The RXFB_NOIO modifier suppresses the inheritance of the
host’s input and output streams.

USAGE:RXREMCON

This code requests that an entry be removed from the Clip List. Parameter slot ARGO points to the null-terminated name to be removed. The Clip List is searched for a node matching the supplied name,and if a match is found the list node is removed and recycled. If no match is found the packet is returned with a warning error code.

USAGE:RXREMLIB

This command removes a Library List entry. Parameter slot ARGO points to the null terminated string specifying the library to be removed. The Library List is searched for a node matching the library name,and if a match is found the node is removed and released. If no match is found the packet is returned with a warning error code. The libary node will not be removed if the library is currently being used by an ARexx program.

USAGE:RXTCCLS

This code requests that the global tracing console be closed. The console window will be closed immediately unless one or more ARexx programs are waiting for input from the console. In this event,the window will be closed as soon as the active programs are no longer using it.

USAGE:RXTCOPN

This command requests that the global tracing console be opened. Once the console is open,all active ARexx programs will divert their tracing output to the console. Tracing input(for interactive debugging)will also be diverted to the new console. Only one console can be opened;subsequent RXTCOPN requests will be returned with a warning error message.

MODIFIER FLAGS

Command codes may include modifier flags to select various processing options. Modifier flags are specific to certain commands,and are ignored otherwise.

RXFF_NOIO.

This modifier is used with the RXCOMM and RXFUNC command codes to suppress the automatic inheritance of the host’s input and output streams.

RXFF_NONRET.

Specifies that the message packet is to be recycled by the resident process rather than being returned to the sender. This implies tht the sender doesn’t care about whether the requested action succeeded,since the returned packet provides the only means of acknowledgement. (RXFF_NONRET MUST NOT BE USED AT ANY TIME)

RXFF_RESULT.

This modifer is valid with the RXCOMM and RXFUNC commands,and requests that the called program return a result string. If the program EXITs(or RETURNs)with an expression,the expression result is returned to the caller as an argstring. This ArgString then becomes the callers responsibility to release. This is automatically accomplished by using GetResultString(). It is therefore imperitive that if you use RXFF_RESULT then you must use GetResultString() when the message packet is returned to you or you will incure a memory loss equal to the size of the ArgString Structure.

RXFF_STRING.

This modifer is valid with the RXCOMM and RXFUNC command codes. It indicates that the command or function argument(in slot ARGO)is a "string file" rather than a file name.

RXFF_TOKEN.

This flag is used with the RXCOMM code to request that the command string be completely tokenized before being passed to the invoked program. Programs invoked as commands normally have only a single argument string. The tokenization process uses "white space" to separate the tokens,except within quoted strings. Quoted strings can use either single or double quotes,and the end of the command string(a null character) is considered as an implicit closing quote.

EXAMPLES:

```
Port.l=OpenRexxPort("TestPort")
  If Port = NULL End:EndIf
msg.l=CreateRexxMsg(Port,"vc","TestPort")
  If msg=NULL End:EndIf
SendRexxCommand msg,"open",#RXCOMM|#RXFF_RESULT
```

1.9 AREXXLIB

Statement: ReplyRexxMsg

Modes :
Syntax : ReplyRexxMsg rexxmsg,Result1,Result2,ResultString

MODES:AMIGA

DESCRIPTION:

When ARexx sends you a RexxMsg (Other than a reply to yours i.e. sending yours back to you with results) you must repl to the message before ARexx will continue or free that memory associated with that RexxMsg. ReplyRexxMsg accomplishes this for you. ReplyRexxMsg also will only reply to message that requires a reply so you do not have to include message checking routines in your source simply call ReplyRexxMsg on every message you receive wether it is a command or not.

The arguments are;

rexxmsg is the LONGWORD address of the RexxMsg Arexx sent you as returned by GetMsg_(Port).

Result1 is 0 or a severity value if there was an error.

Result2 is 0 or an Arexx error number if there was an error processing the command that was contained in the message.

ResultString is the result string to be sent back to Arexx. This will only be sent if Arexx requested one and Result1 and 2 are 0.

```
ReplyRexxMsg rexxmsg,0,0,"THE RETURNED MESSAGE"
```

1.10 AREXXLIB

```
Statement: GetRexxResult
--------------------------------------------------------------------------------
Modes  :
Syntax : GetRexxResult Result=GetRexxResult(rexxmsg,ResultNum)

MODES:AMIGA

DESCRIPTION:

GetRexxResult extracts either of the two result numbers from the RexxMsg
structure. Care must be taken with this Function to ascertain wether you are
dealing with error codes or a ResultString address. Basically if result 1 is
zero then result 2 will either be zero or contain a ArgString pointer to the
ResultString. This should then be obtained using GetResultString().

The arguments to GetRexxResult are;

rexxmsg is the LONGWORD address of a RexxMsg structure returned from ARexx.

ResultNum is either 1 or 2 depending on wether you wish to check result 1 or
result 2.

;print the severity code if there was an error

NPrint GetRexxResult(msg,1)

;check for ResultString and get it if there is one

IF GetRexxResult(msg,1)=0
IF GetRexxResult(msg,2) THEN GetResultString(msg)
ENDIF
```

1.11 AREXXLIB

```
Statement: GetRexxCommand
--------------------------------------------------------------------------------
Modes  :
Syntax : GetRexxCommand String=GetRexxCommand(rexxmsg,ARGNum)

MODES:AMIGA

DESCRIPTION:

GetRexxCommand allows you access to all 16 ArgString slots in the given
RexxMsg. Slot 1 contains the command string sent by ARexx in a command
```

```
message so this allows you to extract the Command.

Arguments are:

rexxmsg is a LONGWORD address of the RexxMsg structure as returned by
RexxEvent()

ARGNum is an integer from 1 to 16 specifying the ArgString Slot you wish to
get an ArgString from.

BEWARE YOU MUST KNOW THAT THERE IS AN ARGSTRING THERE.
```

1.12 AREXXLIB

```
Statement: GetResultString
--------------------------------------------------------------------------------
Modes  :
Syntax : GetResultString String=GetResultString(rexxmsg)

MODES:AMIGA

DESCRIPTION:

GetResultString allows you to extract the result string returned to you by
ARexx after it has completed the action you requested. ARexx will only send
back a result string if you asked for one (using the ActionCodes) and the
requested action was successful.

;check for ResultString and get it if there is one

IF GetRexxResult(msg,1)=0
 IF GetRexxResult(msg,2) THEN GetResultString(msg)
ENDIF

NOTE: Do not attempt to DeleteArgString the result string returned by this
function as the return is a string and not an ArgString pointer. BB2 will
automatically delete this argstring for you.
```

1.13 AREXXLIB

```
Statement: RexxEvent
--------------------------------------------------------------------------------
Modes  :
Syntax : RexxEvent rexxmsg=RexxEvent(PortAddress)

MODES:AMIGA

DESCRIPTION:

RexxEvent is our Arexx Equivalent of EVENT(). It's purpose is to check the
```

given Port to see if there is a message waiting there for us.

It should be called after a WAIT and will either return a NULL to us if there was no message or the LONG address of a RexxMsg Structure if there was a message waiting.

Multiple Arexx MsgPorts can be handled using separate calls to RexxEvent():

```
Wait:Rmsg1.l=RexxEvent(Port1):Rmsg2.l=RexxEvent(Port2):etc
```

RexxEvent also takes care of automatically clearing the rexxmsg if it is our message being returned to us.

The argument is the LONG address of a MsgPort as returned by CreateMsgPort().

EXAMPLES:

```
Repeat
 Wait:Rmsg.l=REXXEVENT(Port):ev.l=EVENT
 IF IsRexxMsg(Rmsg) Process_Rexx_Messages{}:ENDIF
 ;
 ;
 ;Rest of normal intuition event loop statements case etc
Until ev =$200
```

1.14 AREXXLIB

```
Statement: IsRexxMsg
--------------------------------------------------------------------------------
Modes  :
Syntax : IsRexxMsg Boolean=IsRexxMsg(rexxmsg)
```

MODES:AMIGA

DESCRIPTION:

IsRexxMsg tests the argument (a LONGWORD pointer hopefully to a message packet) to see if it is a RexxMsg Packet. If it is TRUE is returned (1) or FALSE if it is not (0).

```
Repeat
 Wait:Rmsg.l=REXXEVENT:ev.l=EVENT
 IF IsRexxMsg(Rmsg) Process_Rexx_Messages{}:ENDIF
 ;
 ;
;Rest of normal intuition event loop statements case etc
Until ev =$200
```

As the test is non destructive and extensive passing a NULL value or a LONGWORD that does not point to a Message structure (Intuition or Arexx) will safely return as FALSE.

1.15 AREXXLIB

```
Statement: RexxError
--------------------------------------------------------------------------------
Modes  :
Syntax : RexxError ErrorString=RexxError(ErrorCode)

MODES:AMIGA

DESCRIPTION:

RexxError converts a numerical error code such as you would get from
GetRexxResult(msg,2) into an understandable string error message. If the
ErrorCode is not known to ARexx a string stating so is returned this ensures
that this function will always succeed.

NPRINT RexxError(5)
```

1.16 AREXXLIB

```
                .--------------------------------------------------------------------------
|                                  AREXXLIB                                       |
`-----------------------------------------------------------------------------'
```

RexxEvent

SendRexxCommand