ARx_Error.ag

COLLABORATORS

	TITLE : ARx_Error.ag	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 17, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Error.ag

1.1 ARexxGuide | Error codes

```
                  AN AMIGAGUIDE® TO ARexx                  Second edition (v2.0)
by Robin Evans


               About this section
                 Error codes:

               1

               2

               3

               4

               5

               6

               7

               8

               9

               10

               11

               12

               13

               14

               15

               16
```

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

```
                45

                46

                47

                48

                Alphabetic list of error messages
                          Copyright © 1993,1994 Robin Evans.  All rights  ↩
                             reserved.
```


1.2 ARexxGuide | Error codes | ABOUT THIS SECTION

```
                Error codes and messages
~~~~~~~~~~~~~~~~~~~~~~~~
When a syntax error occurs in an ARexx script, the  interpreter  will
print a message listing an error number and an error message. Although a
comprehensive outline is impossible, this section explains some of the
conditions that can trigger each error. Each possible error number is
listed on the main page. The messages printed with the error numbers are
also
                listed alphabetically
               .

The error message is printed to the  STDOUT  device or to  STDERR  if that
device is defined. In some environments, STDOUT is redirected to  NIL: ,
which will send ARexx error messages off to the Amiga-ether. To view the
messages under any condition, a trace console can be opened with the  TCO
command.

An all-too-frequent error message that isn't listed here is one that looks
something like this:

   Unknown command 1
     1 *-* open(t,'t:test',w);
   +++ Command returned 20

That error message was not generated by ARexx, which executed the  OPEN()
function without a problem, but by the operating system. The problem here
is that an  expression  -- the open() function -- was used without making
it a part of a valid ARexx  instruction  or  assignment .

         More information:  Avoiding accidental commands

   Also see  TRACE instruction
             TRACE() function
             TS command
   Tutorial  Debugging a macro

 Compatibility issues:
   The error messages used in ARexx are similar to those defined in
```

TRL2 , but not identical. The error code associated with each message is significantly different.

Next, Prev, and Contents: Error codes

1.3 ARexxGuide | Error codes | Alphabetical listing

Arithmetic conversion error	: Error 47
Boolean value not 0 or 1	: Error 46
Command string error	: Error 11
Error return from function	: Error 12
Execution halted	: Error 2
Expression nesting >32	: Error 43
Expression required	: Error 45
Extraneous characters	: Error 35
Function did not return value	: Error 16
Function not found	: Error 15
Host environment not found	: Error 13
Incomplete IF or SELECT	: Error 29
Insufficient memory	: Error 3
Invalid argument to function	: Error 18
Invalid DO syntax	: Error 28
Invalid expression	: Error 41
Invalid expression result	: Error 44
Invalid keyword	: Error 33
Invalid message packet	: Error 10
Invalid operand	: Error 48
Invalid PROCEDURE	: Error 19
Invalid statement in SELECT	: Error 23
Invalid template	: Error 37

```
Invalid variable name               : Error 40

Keyword conflict                    : Error 36

Label not found                     : Error 30

Missing or multiple THEN            : Error 24

Missing or unexpected END           : Error 26

Missing OTHERWISE                   : Error 25

Program not found                   : Error  1

Requested library not found         : Error 14

Required keyword missing            : Error 34

Symbol expected                     : Error 31

Symbol mismatch                     : Error 27

Symbol or string >65535 characters  : Error  9

Symbol or string expected           : Error 32

Unbalanced parentheses              : Error 42

Unexpected BREAK, LEAVE or ITERATE  : Error 22

Unexpected ELSE or OTHERWISE        : Error 21

Unexpected THEN or WHEN             : Error 20

Uninitialized variable              : Error 39

Unmatched quote                     : Error  5

Unrecognized token                  : Error  8

Unterminated comment                : Error  6

Wrong number of arguments           : Error 17
Next, Prev. & Contents: Error codes
```

1.4 ARexxGuide | Error codes (1 of 48) | Error 1

```
Error 1: Program not found

ARexx follows its own  search path  when looking for a program. If a
script is launched with the command "rx myprog", ARexx will first look for
the file "myprog" in the current directory and then for the file
"myprog.rexx". If neither file is present in the current directory, the
interpreter will search for "rexx:myprog" and then "rexx:myprog.rexx".
```

This error is sometimes caused by lack of opening comment marker in an otherwise valid script. ARexx will not recognize a program as a REXX script unless the first non-space characters in the file are "/*".

1.5 ARexxGuide | Error codes (2 of 48) | Error 2

Error 2: Execution halted

Triggered by a BREAK C interrupt or an external halt signal from HI .

1.6 ARexxGuide | Error codes (3 of 48) | Error 3

Error 3: Insufficient memory

The interpreter was unable to allocate the minimum memory required for its initial scan of the script.

1.7 ARexxGuide | Error codes (4 of 48) | Error code 4

Error code 4 is not used.

1.8 ARexxGuide | Error codes (5 of 48) | Error 5

Error 5: Unmatched quote

The interpreter finds many unmatched quotation marks during its initial scan of a script, before execution starts.

The error will sometimes go unnoticed becuase ARexx allows string tokens to span multiple physical lines. If two quotation marks are missing from a script, the interpreter will often pair the first unterminated mark with the second one, producing an unintentional string spanning many lines of program code.

Also see:

Error 9
Also see:
Error 26

1.9 ARexxGuide | Error codes (6 of 48) | Error 6

```
Error 6: Unterminated comment

This error is recognized during the initial scan of a program.

 Comments  may be nested within other comments, but each "/*" marker must
be paired with a "*/" marker.
```

1.10 ARexxGuide | Error codes (7 of 48) | Error code 7

```
   Error code 7 is not used.
```

1.11 ARexxGuide | Error codes (8 of 48) | Error 8

```
Error 8: Unrecognized token

This error is recognized during the initial scan of a program.

It occurs when a character that is not valid for use as a  symbol ,
 operator , or  special character  is used outside of a  literal string .

The error sometimes occurs when  commands  that use unique characters are
listed without quotation marks.
```

1.12 ARexxGuide | Error codes (9 of 48) | Error 9

```
                Error 9: Symbol or string >65535 characters

 Strings ,  symbols , and  variables  are limited to 64k bytes. This error
will result from an attempt to create a string or (less likely) a symbol
that is longer than that.

The error is sometimes generated by misplaced quotation marks.
                 Error 5
                is signalled if there are an uneven number of quotation marks in a ↩
                    file,
but if two marks are missing, the interpreter will attempt include
everything between the first and second marks in one long string.
```

1.13 ARexxGuide | Error codes (10 of 48) | Error 10

```
                Error 10: Invalid message packet

The  message port functions  usually generate
                 Error 18
                 when an invalid
address is used as an argument. The error may result, though, if the
address is valid, but the returned packet does not meet specifications.

The error code might also be used by  library function  packages.
```

1.14 ARexxGuide | Error codes (11 of 48) | Error 11

```
Error 11: Command string error

This error is not generated under any known condition by ARexx itself, but
might be used by programs that include an ARexx port. The error would
indicate that the command was successfully sent to the  host  but rejected
in that environment. Consult the documentation for the program to which
the command was sent.

The ARexx  interpreter  does not check for valid  command  strings, but
rather sends the command to the  current host  and generates a
 command error  or  command failure  if the host does not recognize the
string.
```

1.15 ARexxGuide | Error codes (12 of 48) | Error 12

```
Error 12: Error return from function

If an error occurs in a function, ARexx will still attempt to return
control to the calling environment, but with an error condition set.

If the error was caused by an  internal  or  external function  then the
original (and more informative) error message will often be printed to the
error console before this message is output.
```

1.16 ARexxGuide | Error codes (13 of 48) | Error 13

```
Error 13: Host environment not found

   "address foo; 'jump hoops'
   +++ Error 13 in line 1: Host environment not found
```

The error indicates that the current host is not available. The error will be associated with a command clause rather than with the ADDRESS instruction that is the real source of the problem.

The ADDRESS instruction changes the current host for a script, but does not verify that the host is available.

This error can be caused by a misspelled name used as the argument to ADDRESS. Host names are case-sensitive and must refer either to "COMMAND" or to a message port open and available on the system at the time the command is issued. Port names that include lower-case letters must be enclosed in quotation marks.

To view available ports from a shell, the following ARexx clause can be used:

```
rx "say show('P',,'0a'x)
```

The AmigaDOS command WaitForPort will delay a script until the proper port has been established by an external program.

1.17 ARexxGuide | Error codes (14 of 48) | Error 14

Error 14: Requested library not found

```
"call addlib('NoLib',0,-30,0); say foo()
+++ Error 14 in line 1: Requested library not found
```

The error is triggered when one of the libraries on the interpreter's resource list could not be loaded. This error will be associated with a function call rather than with the ADDLIB() function that is the real source of the problem.

This error can occur in a different script since library names remain on the list even after the program that put them there exits. Use the AmigaDOS command RXLIB or the instruction 'SAY SHOW('L') ' to list ARexx libraries. Compare the listed libraries to a listing of the libs: directory. Library names are case-sensitive.

Remove incorrect library names with the REMLIB() function.

1.18 ARexxGuide | Error codes (15 of 48) | Error 15

Error 15: Function not found

```
"call 10 + 12
+++ Error 15 in line 1: Function not found

"x= foo()
+++ Error 15 in line 1: Function not found
```

The error is sometimes caused by failure to use the ADDLIB() function or RXLIB command to add an external library to the list of function hosts searched by the interpreter. It might also indicate that an invalid name is included in the library list, since such a name will interfere with valid calls to functions in libraries loaded after the invalid name was added. Remove the invalid name with remlib() to correct the situation.

If an internal function was invoked, a typo in the label would cause this error. Misplaced quotation marks can sometimes cause a label to become invisible to the interpreter . (See
Error 9
.)

If an external function was invoked, this error indicates that the interpreter could not find the program within the ARexx search path . Even if it is called as a function, a script must begin with comment tokens if it is to be recognized.

1.19 ARexxGuide | Error codes (16 of 48) | Error 16

Error 16: Function did not return value

```
"say Func(); exit; Func: return
+++ Error 16 in line 1: Function did not return value
```

Functions must return a value of some kind to the calling environment. The error will be reported if control is passed back to the caller by an internal or external function using RETURN or EXIT without a value.

It is good practice to add at least a "0" to all RETURN and EXIT instructions.

1.20 ARexxGuide | Error codes (17 of 48) | Error 17

Error 17: Wrong number of arguments

```
"say right(34)
+++ Error 17 in line 1: Wrong number of arguments
```

Although functions often include optional arguments that may be omitted, this error is triggered if an argument required by a built-in or library function is missing or if more arguments are included than allowed by the function definition.

See the Functions Reference chapter for arguments templates for each built-in and rexxsupport.library function.

1.21 ARexxGuide | Error codes (18 of 48) | Error 18

```
Error 18: Invalid argument to function

   "say right(2,'word')
   +++ Error 18 in line 1: Invalid argument to function

The error usually occurs when a non-numeric value is used in an argument
slot that requires a  number , but the error might also indicate some
other  datatype  mismatch.
```

1.22 ARexxGuide | Error codes (19 of 48) | Error 19

```
Error 19: Invalid PROCEDURE

   "startover: procedure; say "Begin"
   +++ Error 19 in line 1: Invalid PROCEDURE

The  PROCEDURE  instruction can be used only within a  subroutine . This
error will be triggered if program flow falls through to a subroutine
because of a missing  exit  or  return  instruction in the preceding
section.
```

1.23 ARexxGuide | Error codes (20 of 48) | Error 20

```
Error 20: Unexpected THEN or WHEN

   "if a=a then say 'Yes'; then say 'Equivalent'
   Yes
   +++ Error 20 in line 1: Unexpected THEN or WHEN

The subkeyword  THEN  may be used only within an  IF  or  WHEN
instruction. WHEN may be used only within a SELECT instruction. This error
sometimes indicates that the conditional expression used in the primary
instruction is improperly formed.
```

1.24 ARexxGuide | Error codes (21 of 48) | Error 21

```
Error 21: Unexpected ELSE or OTHERWISE

   "if a=b then say 'Yes'
    say 'Equivalent'
    else say 'Not'

   Equivalent
   +++ Error 21 in line 1: Unexpected ELSE or OTHERWISE
```

The secondary keyword ELSE may be used only within an IF instruction. OTHERWISE may be used only as the final clause in a SELECT instruction. This error will be set if the words appear as keywords in any other situation.

As in the example above, the error is sometimes set when multiple clauses following an IF instruction are not enclosed within a DO/END block.

1.25 ARexxGuide | Error codes (22 of 48) | Error 22

Error 22: Unexpected BREAK, LEAVE or ITERATE

```
"do 2; say 'Hi'; end; iterate;
Hi
Hi
+++ Error 22 in line 1: Unexpected BREAK, LEAVE or ITERATE
```

LEAVE and ITERATE may be used only within an iterative DO loop. BREAK may be used in any DO block or in an INTERPRET statement. Use of the instructions in other situations will cause this error.

1.26 ARexxGuide | Error codes (23 of 48) | Error 23

Error 23: Invalid statement in SELECT

```
"select; say 'Choosing...';
+++ Error 23 in line 1: Invalid statement in SELECT
```

All clauses within a SELECT instruction (between the SELECT keyword and the closing END) must be associated with WHEN or OTHERWISE clause.

This error is often caused by failure to use a DO/END construction with multiple clauses in a WHEN instruction. It might also be generated when instructions typed on the trace console are interpreted as an invalid part of a SELECT instruction.

1.27 ARexxGuide | Error codes (24 of 48) | Error 24

Error 24: Missing or multiple THEN

```
"if foo=boo;
+++ Error 24 in line 1: Missing or multiple THEN

"select;otherwise;end;
+++ Error 24 in line 1: Missing or multiple THEN
```

The conditional expression in each IF or WHEN clause must be followed by the secondary keyword THEN , which is a valid keyword only when associated with IF or WHEN.

An error in the IF conditional is usually trapped as
Error 46
.

1.28 ARexxGuide | Error codes (25 of 48) | Error 25

Error 25: Missing OTHERWISE

```
"select;end;
+++ Error 25 in line 1: Missing  OTHERWISE
```

This error is caught only if all WHEN clauses have been exhausted without a match.

1.29 ARexxGuide | Error codes (26 of 48) | Error 26

Error 26: Missing or unexpected END

```
"say 'Hi'; end;
Hi
+++ Error 26 in line 1: Missing or unexpected END
```

The range of each DO or SELECT instruction must be closed with the END secondary keyword, which is valid only when associated with DO or SELECT.

A DO instruction will enclose all clauses within its range until a matching END is found or until the end of the script. Failure to include an END to a SELECT range will cause the following clauses to be interpreted as part of the OTHERWISE range.

This error will often occur at the end of a script, indicating that a larger-than-intended range of clauses was included with one of the block instructions.

The unmatched DO or SELECT can often be found by stepping backwards in a TRACE output. For that purpose, the minimal output of TRACE A is usually adequate.

A misplaced quotation mark that is not trapped by
error 5
or
error 9
is sometimes indicated by this error. That happens when the ↩
opening DO or

SELECT keyword or the closing END is enclosed in the spanning literal
string and therefore invisible. A TRACE of the code will show missing
lines of code or will show a string that includes what should be clauses.

1.30 ARexxGuide | Error codes (27 of 48) | Error 27

Error 27: Symbol mismatch

This error is infrequently generated by ARexx, but may be triggered by a
 library function. It indicates that a symbol used in a clause is not
of the expected type.

1.31 ARexxGuide | Error codes (28 of 48) | Error 28

Error 28: Invalid DO syntax

```
   "do 5 to 6
   +++ Error 28 in line 1: Invalid DO syntax
```

The error is often generated by failure to use an index variable when
the TO subkeyword is used in a DO instruction, but other errors in
a DO instruction will also trigger this error.

1.32 ARexxGuide | Error codes (29 of 48) | Error 29

Error 29: Incomplete IF or SELECT

```
   "if foo=boo then;
   +++ Error 29 in line 1: Incomplete IF or SELECT
```

The error usually indicates that a clause is not bound to a THEN
keyword. Failure to include a conditional expression will be trapped by

Error 45
.

During program development, it may be useful to include an incomplete IF
instruction. That can be done by using the NOP keyword as the THEN
clause or by using an empty DO/END block:

```
   Example:
      if foo=boo then nop;
   or
      if foo=boo then do;end;
```

1.33 ARexxGuide | Error codes (30 of 48) | Error 30

```
               Error 30: Label not found

   signal foo
   +++ Error 30 in line 1: Label not found
```

The error is almost always generated by a SIGNAL instruction, since invalid function calls are discovered by other means and generate

```
                Error 15
               .
```

When any of the SIGNAL instructions are used, a label matching the signal's name must be included within the script.

1.34 ARexxGuide | Error codes (31 of 48) | Error 31

```
               Error 31: Symbol expected
```

The error occurs when a symbol required by an instruction is missing. It is generated only if the required argument must be a symbol. If the argument could be a string or a keyword , then the fault triggered would be

```
                Error 32
               ,
                Error 33
               , or
                Error 34
               .
```

1.35 ARexxGuide | Error codes (32 of 48) | Error 32

```
               Error 32: Symbol or string expected

   "call
   +++ Error 32 in line 1: Symbol or string expected
```

Often indicates an incomplete clause . Several keywords require an expression and cannot be used in isolation.

```
                Error 45
                is the more
```

general form of this error. Error 32 indicates that the missing element must be either a string or a symbol .

1.36 ARexxGuide | Error codes (33 of 48) | Error 33

```
Error 33: Invalid keyword
```

The keyword used in the instruction is invalid in this context.

1.37 ARexxGuide | Error codes (34 of 48) | Error 34

```
Error 34: Required keyword missing
```

```
   "parse value 'Hi world.' Greeting Audience
   +++ Error 34 in line 1: Required keyword missing
   "signal on;
   +++ Error 34 in line 1: Required keyword missing
   "numeric;
   +++ Error 34 in line 1: Required keyword missing
```

Some instructions require a subsidiary keyword in certain situations. This error is generated when one of the expected subkeywords is missing. It cannot be generated by failure to use a primary keyword since such a clause would be interpreted as a command .

This error is popular with the VALUE option to PARSE since that option requires a sub-subkeyword, WITH. It may also be generated by the NUMERIC instruction which requires the VALUE subkeyword if a variable is used to indicate the setting.

1.38 ARexxGuide | Error codes (35 of 48) | Error 35

```
Error 35: Extraneous characters
```

```
   options 'Enter a value: '
   +++ Error 35 in line 1: Extraneous characters
   select foo;
   +++ Error 35 in line 1: Extraneous characters
   signal on break_c:;
   +++ Error 35 in line 1: Extraneous characters
```

This error may indicates a misuse of keywords and subkeywords.

In the first example above, the error is generated by failure to include the required subkeyword, PROMPT , with the OPTIONS instruction. In the second example, an expression is included after the SELECT keyword which must stand alone in a clause. (Future extensions to REXX may allow use of an expression with SELECT, but it is not yet a valid syntax.) The error in the third line is generated by the `:' following the error-trap name. Colons may be used only as part of a label .

The error will also occur when a token that is otherwise valid is used in an improper context:

```
   say  copies('-', PrdWd-1), copies('-', CdWd-1)
   +++ Error 35 in line 13: Extraneous characters
```

The extra comma after the first copies() function would be valid as a continuation token if the line were split into two parts at that point, but is not valid in this circumstance.

1.39 ARexxGuide | Error codes (36 of 48) | Error 36

```
Error 36: Keyword conflict
```

Two or more keywords that are valid in isolation are combined in an invalid way in the instruction .

1.40 ARexxGuide | Error codes (37 of 48) | Error 37

```
Error 37: Invalid template
```

```
   "parse var;
   +++ Error 37 in line 1: Invalid template
```

The error is always associated with a PARSE instruction. The various source-string options impose different requirements on the template . In the example above, for instance, a error results from failure to specify a template with the VAR source string. It is, on the other hand, valid to omit the template with the PULL source string keyword.

The error may also result from improper use of template variables .

1.41 ARexxGuide | Error codes (38 of 48) | Error code 38

```
   Error code 38 is not used.
```

1.42 ARexxGuide | Error codes (39 of 48) | Error 39

```
Error 39: Uninitialized variable
```

This error is not generated by ARexx under any known condition. Use of uninitialized variable is not normally an error. Even when the SIGNAL ON NOVALUE trap is set, the interpreter passes control to the NOVALUE: subroutine without setting this error.

The user code within a NOVALUE subroutine, or an external function library might make use of this error code and message.

1.43 ARexxGuide | Error codes (40 of 48) | Error 40

Error 40: Invalid variable name

This error is rarely generated because the interpreter will treat the value in some other (and probably unintended) way.

‘3var = "Hi"’ is an invalid assignment clause since ‘3Var’ is not a valid variable name , but the interpreter will not trap the error: The clause would be evaluated as a logical expression . The result, 0, would be sent as a command string to the default host.

Other situations where an invalid variable name are used are similarly hidden under different errors:

```
"do 3boo = 7 to 9; say 3boo; end
+++ Error 28 in line 1: Invalid DO syntax

"parse value 'hi there' with 6greet 6who
+++ Error 47 in line 1: Arithmetic conversion error
```

1.44 ARexxGuide | Error codes (41 of 48) | Error 41

Error 41: Invalid expression

```
"say 2+3+;
+++ Error 41 in line 1: Invalid expression
say exists(help:arx/arx_elements.ag)
+++ Error 41 in line 1: Invalid expression
```

This error is often associated with an invalid use of operator characters.

The characters representing operators are reserved : They can be used only for their defined purposes and are invalid in other contexts. This sometimes presents a problem when commands are used without enclosing quotation marks. If operator characters (like the ‘/’ used in Amiga file names) are included within a clause, ARexx will attempt to perform the indicated operation and generate error 41 in the process.

The second error occurred because the AmigaDOS file name uses the division character. To avoid such errors, quotation marks should enclose all strings -- including command clauses -- that are not meant to be evaluated.

Also see

Error 47

1.45 ARexxGuide | Error codes (42 of 48) | Error 42

```
                Error 42: Unbalanced parentheses

   say right(trunc(Num,2),6
   +++ Error 42 in line 1: Unbalanced parentheses
```

Each opening parenthesis in ARexx must be paired with a closing parenthesis. Unlike
Error 5
, this error is detected only when ARexx attempts to interpret the offending expression.

1.46 ARexxGuide | Error codes (43 of 48) | Error 43

```
Error 43: Expression nesting >32
```

A maximum of 32 expressions may be nested in a single clause . This limit applies only to nested expressions and not to the number of terms that can be used in a single expression.

It is valid, for instance, to add hundreds of numbers or to concatenate hundreds of strings in a single expression. This error occurs only when subexpressions (usually indicated by parentheses) are nested within other expressions.

The error might be caused by a complex set of functions that used the results of other functions as arguments

1.47 ARexxGuide | Error codes (44 of 48) | Error 44

```
Error 44: Invalid expression result

   "numeric fuzz 10;
   +++ Error 44 in line 1: Invalid expression result
```

The error is triggered when an otherwise valid expression yields a result that is invalid in the current context.

In the example above, the expression, ‘10’, used with NUMERIC FUZZ is greater than the current setting of NUMERIC DIGITS, which is not allowed.

1.48 ARexxGuide | Error codes (45 of 48) | Error 45

```
                Error 45: Expression required

   "if;
   +++ Error 45 in line 1: Expression required
   "signal;
   +++ Error 45 in line 1: Expression required
```

The error is triggered when an instruction requires further information that is not provided.

A conditional expression must be used with IF and WHEN instructions. This error will result if there is no expression of any kind.

```
                Error 46
                results when an expression is included, but it does not yield a   ↩
                   Boolean
value.
```

1.49 ARexxGuide | Error codes (46 of 48) | Error 46

```
Error 46: Boolean value not 0 or 1

   "if boo then say hi
   +++ Error 46 in line 1: Boolean value not 0 or 1
   "say boo & foo
   +++ Error 46 in line 1: Boolean value not 0 or 1
```

The error often occurs when an expression used with a conditional statement, such as IF or WHEN , is not properly formatted. Only a conditional expression is valid.

Some functions, such as SHOW() and ARG() , can be used as conditional expressions, but only if the proper arguments are supplied.

1.50 ARexxGuide | Error codes (47 of 48) | Error 47

```
Error 47: Arithmetic conversion error

   "say boo + foo
   +++ Error 47 in line 1: Arithmetic conversion error
```

This is a common error in ARexx since the language treats all values as strings until they are used in an arithmetic operation . If a value used in an arithmetic operation cannot be translated to a valid number , this error is generated.

The error may be triggered by failure to use quotation marks in a filename since the "/" character in Amiga filenames is the division operator in ARexx. ARexx will treat the two parts of the filename as variables and

```
attempt to divide them. It generates this error because the variables
could not be interpreted as valid number.

   Also see:  Basic elements: Numbers as text
```

1.51 ARexxGuide | Error codes (48 of 48) | Error 48

```
               Error 48: Invalid operand

   "say 3/0
   +++ Error 48 in line 1: Invalid operand

The error usually indicates a divide-by-zero error. Most arithmetic errors
other than divide-by-zero will be trapped as
                Error 47
               .
```