```
/* Classic REXX 5.00 (Regina) or 6.03+ (ooRexx) with RexxUtil     */

   signal on novalue  name ERROR ;  parse version UTIL REXX .
   if ( 0 <> x2c( 30 )) | ( REXX <> 5 & REXX < 6.03 )
      then  exit ERROR( 'untested' UTIL REXX )
   if 6 <= REXX   then  interpret  'signal on nostring   name ERROR'
   if 5 <= REXX   then  interpret  'signal on lostdigits name ERROR'
   signal on halt     name ERROR ;  signal on failure    name ERROR
   signal on notready name ERROR ;  signal on error      name ERROR
   numeric digits 20             ;  UTIL = REGUTIL()

/* ------------------------------------------------------------- */

   parse arg STR                 ;  TMP = '? -? /? -h /h'
   if arg() <> 1 | STR = ''      then  exit USAGE()
   if sign( wordpos( STR, TMP )) then  exit USAGE()

   TMP = time( 'r' )             ;  signal off error
   if REXX == 5.00   then  address SYSTEM STR
                     else  address CMD    STR
   say time( 'e' )               ;  return rc

/* ---------------------------- (REXX USAGE template 2016-03-06) */

USAGE:   procedure               /* show (error +) usage message: */
   parse source . . USE          ;  USE = filespec( 'name', USE )
   say x2c( right( 7, arg()))    /* terminate line (BEL if error) */
   if arg() then  say 'Error:' arg( 1 )
   say 'Usage:' USE 'cmdline'
   say                           /* suited for REXXC tokenization */
   say " Shows time('elapsed') after running ADDRESS CMD cmdline."
   return 1                      /* exit code 1, nothing happened */

/* ---------------------------- (Regina SysLoadFuncs 2015-12-06) */

REGUTIL: procedure               /* Not needed for ooRexx > 6.03  */
   if RxFuncQuery( 'SysLoadFuncs' ) then  do
      ERR = RxFuncAdd( 'SysLoadFuncs', 'RexxUtil' )
      if ERR <> 0 then  exit ERROR( 'RexxUtil load error' ERR )
   end                           /* static Regina has no RexxUtil */
   ERR = SysLoadFuncs()          ;  return SysUtilVersion()

/* ---------------------------- (STDERR: unification 2020-03-14) */
/* PERROR() emulates lineout( 'STDERR:', emsg ) with ERROUT().    */
/* ERROUT() emulates charout( 'STDERR:', emsg ).                  */

/* ERROR() shows an error message and the source line number sigl */
/* on stderr.  Examples:   if 0 = 1 then  exit ERROR( 'oops' )    */
/*                         call ERROR 'interactive debug here'    */

/* ERROR() can also catch exceptions (REXX conditions), examples: */
/* SIGNAL ON ERROR               non-zero rc or unhandled FAILURE */
/* SIGNAL ON NOVALUE NAME ERROR  uninitialized variable           */
/* CALL ON NOTREADY NAME ERROR   blocked I/O (incl. EOF on input) */

/* ERROR() uses ERROR. in the context of its caller and returns 1 */
/* for explicit calls or CALL ON conditions.  For a SIGNAL ON ... */
```

```
/* condition ERROR() ends with exit 1.                              */

PERROR:  return sign( ERROUT( arg( 1 ) || x2c( 0D0A )))
ERROUT:  procedure
   parse version S V .           ;  signal off notready
   select
      when  6 <= V & V < 7 then  S = 'STDERR:'        /* (o)oRexx */
      when  S == 'REXXSAA' then  S = 'STDERR:'        /* IBM Rexx */
      when  V == 5.00      then  S = '<STDERR>'       /* Regina   */
      otherwise                  S = '/dev/con'       /* Quercus  */
   end                           /* Kedit KEXX 5.xy not supported */
   return charout( S, arg( 1 ))

ERROR:                           /* trace off, save result + sigl */
   ERROR.3 = trace( 'o' )        ;  ERROR.1 = value( 'result' )
   ERROR.2 = sigl                ;  call PERROR
   ERROR.3 = right( ERROR.2 '*-*', 10 )
   if ERROR.2 <= sourceline()
      then  call PERROR ERROR.3 strip( sourceline( ERROR.2 ))
      else  call PERROR ERROR.3 '(source line unavailable)'

   ERROR.3 = right( '+++', 10 ) condition( 'c' ) condition( 'd' )
   if condition() = ''  then  ERROR.3 = right( '>>>', 10 ) arg( 1 )
   call PERROR ERROR.3
   select
      when  sign( wordpos( condition( 'c' ), 'ERROR FAILURE' ))
      then  ERROR.3 = 'RC' rc
      when  condition( 'c' ) = 'SYNTAX'
      then  ERROR.3 = errortext( rc )
      when  condition( 'c' ) = 'HALT'
      then  ERROR.3 = errortext( 4 )
      when  condition( 'c' ) = 'NOTREADY' then  do
         ERROR.3 = condition( 'd' )
         if ERROR.3 <> ''  then  do
            ERROR.3 = stream( ERROR.3, 'd' )
         end
      end
      otherwise   ERROR.3 = ''
   end
   if ERROR.3 <> ''  then  call PERROR right( '>>>', 10 ) ERROR.3
   parse value ERROR.2 ERROR.1 with sigl result
   if ERROR.1 == 'RESULT'  then  drop result
   trace ?L                      /* -- interactive label trace -- */
ERROR:   if condition() = 'SIGNAL'  then  exit 1
                                    else  return 1
```