2016 RexxLA International Rexx Language Symposium Proceedings

René Vincent Jansen (ed.)

THE REXX LANGUAGE ASSOCIATION
RexxLA Symposium Proceedings Series
ISSN 1534-8954

Publication Data

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2016 edition is registered under ISBN 978-94-648-5991-1

2023-05-31 First printing

Introduction

History of the International Rexx Language Symposium

In 1990, Cathie Dager of SLAC[1] convened the organizing committee for the first independent Rexx[2] Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the RexxLA took over that responsibility. Symposia have been held annually since 1990.

About RexxLA

During the 1993 Symposium in La Jolla, California, plans for a Rexx User Group materialized. The Rexx Language Association (RexxLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. RexxLA manages several open source implementations of Rexx.

The selection procedure

Presentation proposals are solicited yearly using a CFP[3] procedure, after which the RexxLA symposium comittee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

Location

The 2016 symposium was held in Tampa, Florida, USA from 28 Aug 2016 to 31 Aug 2016.

[1] Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory
[2] Cowlishaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.
[3] Call For Papers.

Contents

1

Open Object Rexx Tutorial – Rony G. Flatscher

Date and Time

28 Aug 2016, 20:30:00 CET

Presenter

Rony G. Flatscher

Presenter Details

Rony works as a professor for Business informatics ("Wirtschaftsinformatik") at the Vienna University of Economics and Business Administration (Wirtschaftsuniversität Wien) and uses Open Object Rexx for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooRexx, the ooRexx-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

Session Abstract

This tutorial introduces ("classic") Rexx programmers to new features ooRexx makes available, which make Rexx programming even easier. It concludes with introducing and demonstrating the creation and usage of Rexx classes in ooRexx, which is very easy, yet powerful. With the proliferation of ooRexx on many platforms, including IBM mainframes, classic Rexx programmers will benefit greatly from this tutorial.

"Leaping from Classic to Object"

2016 International Rexx Symposium

Tampa, Florida

(August 2016)

© 2016 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (http://www.wu.ac.at)

Agenda

- History
- Getting Object Rexx
- New procedural features
- New object-oriented features
- Roundup

2

History, 1

- Begin of the 90s
  - OO-version of Rexx presented to the IBM user group "SHARE"
  - Developed since the beginning of the 90'ies
  - 1997 Introduced with OS/2 Warp 4
    - *Support of SOM and WPS*
  - 1998 Free Linux version, trial version for AIX
  - 1998 Windows 95 and Windows/NT

History, 2

- 2004
  - Spring: RexxLA and IBM join in negotiations about opensourcing Object REXX
  - November: RexxLA gets sources from IBM
  - Opensource developers taking responsibility
    - David Ashley, USA, OS2 guru, Linux freak, ooRexx aficionado
    - Rick McGuire, USA, original lead developer
    - Mark Hessling, Australia, Regina maintainer, author of numerous great, opensource, openplatform Rexx function packages
    - Rony G. Flatscher, Austria (Europe!), author of BSF4Rexx, ooRexx tester of many years
- 2005
  - Spring (March/April): RexxLA makes ooRexx freely available as opensource and openplatform
    - **2005-03-25: ooRexx 3.0**

History, 3

- Summer 2009
  - ooRexx 4.0.0
  - Kernel fully rewritten
    - 32-bit *and* 64-bit versions possible for the first time
    - New OO-APIs into the ooRexx kernel
      - e.g. BSF4ooRexx was created which allows for implementing Java methods in Rexx !
- Latest release
  - August 2016
    - ooRexx 4.2, Feb 24, 2014
    - AIX, Linux, MacOSX, Windows

Getting "Open Object Rexx" ("ooRexx") … for Free!

- http://www.RexxLA.org
  - Choose the link to "ooRexx"
- http://www.ooRexx.org
  - Homepage for ooRexx
  - Links to Sourceforge
    - Source
    - Precompiled versions for AIX, Linux (Debian, K/Ubuntu, Red Hat, Suse,), MacOSX, Solaris, Windows
    - Consolidated (great!) PDF- and HTML-rendered documentation!

New Procedural Features, 1

- Fully compatible with classic Rexx, TRL 2
  - New: execution of a Rexx program
    - *Full syntax check of the Rexx program*
    - *Interpreter carries out all directives (leadin with "**::**")*
    - *Start of program*
- "rexxc.exe": explicit tokenization of Rexx programs
- **USE** ARG in addition to PARSE ARG
  - among other things allows for retrieving stems by reference (!)

Example (ex_stem.rex) "USE ARG" with a Stem

```
/* ex_stem.rex: demonstrating USE ARG  */

info.1 = "Hi, I am a stem which could not get altered in a procedure!"
info.0 = 1              /* indicate one element in stem                          */
call work info.         /* call procedure which adds another element (entry)     */
do i=1 to info.0        /* loop over stem                                        */
   say info.i           /* show content of stem.i                                */
end
exit

work: procedure
   use arg great.       /* note the usage of "USE ARG" instead of "PARSE ARG"    */
   idx = great.0 + 1    /* get number of elements in stem, enlarge it by 1       */
   great.idx = "Object Rexx allows to directly access and manipulate a stem!"
   great.0 = idx        /* indicate new number of elements in stem               */
   return

/* yields:

   Hi, I am a stem which could not get altered in a procedure!
   Object Rexx allows to directly access and manipulate a stem!
*/
```

New Procedural Features, 2

- Routine-directive
  - same as a function/procedure
  - if public, then even callable from another (!) program
- Requires-directive
  - allows for loading programs ("modules") with public routines and public classes one needs
- User definable exceptions

OO-Features Simply Usable by Classic Rexx Programs

- "Environment"
  - a directory object
    - *allows to store data with a key (a string)*
    - *sharing information (coupling of) among different Rexx programs*
  - "**.local**"
    - *available to all Rexx programs within the same Rexx interpreter instance in a process*
  - "**.environment**"
    - *available to all Rexx programs running under all Rexx interpreter instances within the same process*
    - *gets searched after **.local***

Example (dec2roman.rex) Classic Style

```
/* dec2roman.rex: turn decimal number into Roman style   */
Do forever
   call charout "STDOUT:", "Enter a number in the range 1-3999: "; PARSE PULL number
   If number = 0 then exit
   say "    --->" number "=" dec2rom(number)
End

dec2rom: procedure
  PARSE ARG num, bLowerCase     /* mandatory argument: decimal whole number              */
  a.        = ""
        /* 1-9 */      /* 10-90 */    /* 100-900 */   /* 1000-3000 */
  a.1.1  = "i"   ; a.2.1  = "x"   ; a.3.1  = "c"   ; a.4.1  = "m"   ;
  a.1.2  = "ii"  ; a.2.2  = "xx"  ; a.3.2  = "cc"  ; a.4.2  = "mm"  ;
  a.1.3  = "iii" ; a.2.3  = "xxx" ; a.3.3  = "ccc" ; a.4.3  = "mmm" ;
  a.1.4  = "iv"  ; a.2.4  = "xl"  ; a.3.4  = "cd"  ;
  a.1.5  = "v"   ; a.2.5  = "l"   ; a.3.5  = "d"   ;
  a.1.6  = "vi"  ; a.2.6  = "lx"  ; a.3.6  = "dc"  ;
  a.1.7  = "vii" ; a.2.7  = "lxx" ; a.3.7  = "dcc" ;
  a.1.8  = "viii"; a.2.8  = "lxxx"; a.3.8  = "dccc";
  a.1.9  = "ix"  ; a.2.9  = "xc"  ; a.3.9  = "cm"  ;
  IF num < 1 | num > 3999 | \DATATYPE(num, "W") THEN
  DO
     SAY num": not in the range of 1-3999, aborting ..."
     EXIT -1
  END

  num = reverse(strip(num))     /* strip & reverse number to make it easier to loop      */
  tmpString = ""
  DO i = 1 TO LENGTH(num)
     idx = SUBSTR(num,i,1)
     tmpString = a.i.idx || tmpString
  END

  bLowerCase = (translate(left(strip(bLowerCase),1)) = "L")    /* default to uppercase   */
  IF bLowerCase THEN RETURN           tmpString
                ELSE RETURN TRANSLATE(tmpSTring)               /* x-late to uppercase    */
```


Example (routine1_dec2roman.rex)

```
/* routine1_dec2roman.rex: initialization */
  a.        = ""
        /* 1-9 */      /* 10-90 */    /* 100-900 */   /* 1000-3000 */
  a.1.1  = "i"   ; a.2.1  = "x"   ; a.3.1  = "c"   ; a.4.1  = "m"   ;
  a.1.2  = "ii"  ; a.2.2  = "xx"  ; a.3.2  = "cc"  ; a.4.2  = "mm"  ;
  a.1.3  = "iii" ; a.2.3  = "xxx" ; a.3.3  = "ccc" ; a.4.3  = "mmm" ;
  a.1.4  = "iv"  ; a.2.4  = "xl"  ; a.3.4  = "cd"  ;
  a.1.5  = "v"   ; a.2.5  = "l"   ; a.3.5  = "d"   ;
  a.1.6  = "vi"  ; a.2.6  = "lx"  ; a.3.6  = "dc"  ;
  a.1.7  = "vii" ; a.2.7  = "lxx" ; a.3.7  = "dcc" ;
  a.1.8  = "viii"; a.2.8  = "lxxx"; a.3.8  = "dccc";
  a.1.9  = "ix"  ; a.2.9  = "xc"  ; a.3.9  = "cm"  ;
.local~dec.2.rom = a.                 /* save in .local-environment for future use     */

::routine dec2roman public
  PARSE ARG num, bLowerCase           /* mandatory argument: decimal whole number      */

  a. = .local~dec.2.rom               /* retrieve stem from .local-environment         */
  IF num < 1 | num > 3999 | \DATATYPE(num, "W")THEN
  DO
     SAY num": not in the range of 1-3999, aborting ..."
     EXIT -1
  END

  num = reverse(strip(num))     /* strip & reverse number to make it easier to loop     */
  tmpString = ""
  DO i = 1 TO LENGTH(num)
     idx = SUBSTR(num,i,1)
     tmpString = a.i.idx || tmpString
  END

  bLowerCase = (translate(left(strip(bLowerCase),1)) = "L")    /* default to uppercase  */
  IF bLowerCase THEN RETURN           tmpString
                ELSE RETURN TRANSLATE(tmpSTring)               /* x-late to uppercase   */
```

Example (use_routine1_dec2roman.rex)

```
/* use_routine1_dec2roman.rex */
Do forever
   call charout "STDOUT:", "Enter a number in the range 1-3999: "
   PARSE PULL number
   If number = 0 then exit
   say "   --->" number "=" dec2roman(number)
End

::requires "routine1_dec2roman.rex" /* directive to load module with public routine  */
```


Example (routine2_dec2roman.rex)

```
/* routine2_dec2roman.rex: Initialization code  */
 d1    = .array~of( "", "i", "ii", "iii", "iv", "v", "vi", "vii", "viii", "ix" )
 d10   = .array~of( "", "x", "xx", "xxx", "xl", "l", "lx", "lxx", "lxxx", "xc" )
 d100  = .array~of( "", "c", "cc", "ccc", "cd", "d", "dc", "dcc", "dccc", "cm" )
 d1000 = .array~of( "", "m", "mm", "mmm"                                       )
.local~roman.arr = .array~of( d1, d10, d100, d1000 )    /* save in local environment    */

::ROUTINE dec2roman PUBLIC              /* public routine to translate number into Roman*/
  USE ARG num, bLowerCase               /* mandatory argument: decimal whole number     */

  IF num < 1 | num > 3999 | \DATATYPE(num, "W") THEN
     RAISE USER NOT_A_VALID_NUMBER      /* raise user exception                         */

  num = num~strip~reverse       /* strip & reverse number to make it easier to loop     */
  tmpString = ""
  DO i = 1 TO LENGTH(num)
     tmpString = .roman.arr[i] ~at(SUBSTR(num,i,1)+1) || tmpString
  END

  bLowerCase = (bLowerCase~strip~left(1)~translate = "L")       /* default to uppercase */
  IF bLowerCase THEN RETURN           tmpString
                ELSE RETURN TRANSLATE(tmpSTring)                /* x-late to uppercase  */
```

Example (use_routine2_dec2roman.rex)

```
/* use_routine2_dec2roman.rex */
Do forever
   call charout "STDOUT:", "Enter a number in the range 1-3999: "
   PARSE PULL number
   If number = 0 then exit
   say "    --->" number "=" dec2roman(number)
End

::requires "routine2_dec2roman.rex" /* directive to load module with public routine  */
```


New Object-oriented Features, 1

- Allows for implementing abstract data types
  - "Data Type" (DT)
    - *a data type defines the set of valid values*
    - *a data type defines the set of valid operations for it*
    - *examples*
      - *numbers: adding, multiplying, etc*
      - *strings: translating case, concatenating, etc.*
  - "Abstract Data Type" (ADT)
    - *a generic schema defining a data type with*
      - *attributes*
      - *operations on attributes*

New Object-oriented Features, 2

- Object-oriented features of Rexx
  - allow for implementing an ADT
  - a predefined classification tree
  - allow for (multiple) inheritance
  - explicit use of metaclasses
  - tight security manager (!)
    - *allows for implementing any security policy w.r.t. Rexx programs*
      - *untrusted programs from the net*
      - *roaming agents*
      - *company policy w.r.t. executing code in secured environment*

Example (dog.rex) Defining Dogs ...

```
/* dog.rex: a program for dogs ...  */

myDog = .dog~new         /* create a dog from the class          */
myDog~Name = "Sweety"    /* tell the dog what it is called       */
say "My name is:" myDog~Name    /* now ask the dog for its name */
myDog~Bark               /* come on show them who you are!       */

::class  Dog             /* define the class "Dog"               */
::method Name attribute  /* let it have an attribute             */
::method Bark            /* let it be able to bark               */
  say "Woof! Woof! Woof!"

/* yields:

   My name is: Sweety
   Woof! Woof! Woof!

*/
```

Example (bigdog.rex)
Defining BIG Dogs ...

```
/* bgdoc.rex: a program for BIG dogs ...  */

myDog = .BigDog~new      /* create a BIG dog from the class      */
myDog~Name = "Arnie"     /* tell the dog what it is called       */
say "My name is:" myDog~Name    /* now ask the dog for its name */
myDog~Bark               /* come on show them who you are!       */

::class  Dog             /* define the class "Dog"               */
::method Name attribute /* let it have an attribute             */
::method Bark            /* let it be able to bark               */
  say "Woof! Woof! Woof!"

  /* the following class reuses most of what is already
     defined for the class "Dog" via inheritance; it overrides
     the way a big dog barks                                     */
::class  BigDog subclass Dog    /* define the class "BigDog"    */
::method Bark            /* let it be able to bark               */
  say "WOOF! WOOF! WOOF!"

/* yields:

   My name is: Arnie
   WOOF! WOOF! WOOF!

*/
```


New Object-oriented Features, 3

- Object Rexx' classification tree
  - Fundamental classes
    - *Object, Class, Method, Message*
  - Classic Rexx classes
    - *String, Stem, Stream*
  - Collection classes
    - *Array, CircularQueue, List, Queue, Supplier*
    - *Directory, Properties, Relation and Bag, Table, Set*
      - *index is set explicitly by programs*
  - Miscellaneous classes
    - *Alarm, Monitor, ...*

Example (fruit.rex)
A Bag Full of Fruits ...

```
/* fruit.rex: a bag, full of fruits ...   */

Fruit_Bag = .bag~of( "apple", "apple", "pear", "cherry", "apple", "banana",         ,
                     "plum", "plum", "banana", "apple", "pear", "papaya",           ,
                     "peanut", "peanut", "peanut", "peanut", "peanut", "apple",     ,
                     "peanut", "pineapple", "banana", "plum", "pear", "pear",       ,
                     "plum", "plum", "banana", "apple", "pear", "papaya",           ,
                     "peanut", "peanut", "peanut", "apple", "peanut", "pineapple",  ,
                     "banana", "peanut", "peanut", "peanut", "peanut", "peanut",    ,
                     "apple", "peanut", "pineapple", "banana", "peanut", "papaya",  ,
                     "mango", "peanut", "peanut", "apple", "peanut", "pineapple",   ,
                     "banana", "pear" )

SAY "Total of fruits in bag:" Fruit_Bag~items
SAY

Fruit_Set = .set~new~union(Fruit_Bag)
SAY "consisting of:"
DO fruit OVER Fruit_Set
   SAY right(fruit, 21) || ":" RIGHT( Fruit_Bag~allat(fruit)~items, 3 )
END
```


Example (fruit.rex)
Output

```
Total of fruits in bag: 56

consisting of:
                 plum:   5
               cherry:   1
                 pear:   6
                mango:   1
               banana:   7
               peanut:  20
            pineapple:   4
               papaya:   3
                apple:   9
```

Open Object Rexx ("ooRexx") Roundup

- Adds features, long asked for, e.g.
  - Variables (stems) by reference (USE ARG)
  - Public routines available to other programs (concept of modules)
  - Very powerful and complete implementation of the OO-paradigm
- Availability
  - Free
  - Opensource
  - Openplatform
    - Precompiled versions for: AIX, Linux (rpm, deb), MacOSX, Solaris, Windows 98/NT/2000/XP/Vista/W7/W8
- Rony G. Flatscher, *„Introduction to Rexx and ooRexx“,* order form:

  ***h**ttp://www.facultas.at/flatscher*
- TBD*: http://www.RonyRexx.net*

23

2

SHOWIT - An ooRexx tool to collect and display data from stems, files and strings, with substitution. – Les Koehler

Date and Time

29 Aug 2016, 13:30:00 CET

Presenter

Les Koehler

Presenter Details

About the speaker: Les has been involved with Rexx since he received the initial distribution of Mike Cowlishaw's first Specification on 1 May 1979 at the IBM Research Triangle Park Lab just outside Raleigh NC, where they had a VM/370 mainframe and Les developed VM tools and applications.

Session Abstract

Les will present Open Obect Rexx code that makes it easy to create and display data taken from multiple sources: whole files (or pieces of them), stems and literals. Any of these can contain variables that start with an ampresand, much like the html convention for built-in variables. A live demonstration may be done. If time permits, and there is interest, he will explore the coding techniques used.

Contributors

These RexxLA members contributed to this effort:

- Gil Barmwater - ooRexx code and consulting.
- Walter Pachl - Consulting and testing
- Jon Wolfers - Bug resolution code and consulting

I am deeply indebted to all of them.

History

I had written code for The Hessling Editor (THE) macros that helped me develop the processes for handling RexxLA Membershp Applications and Payments. Part of that effort included Help for all the macros, which was extracted from the prolog in the macro itself.

Eventually, that code was moved to a single macro, thus reducing the size of each macro and ensuring consistency.

Then, in October of 2015, I had to write some .rex code and realized I didn't have a similar tool for .rex code as I did for .the code. By the end of the month **showit.rex** was basically functional.

It wasn't until mid July of 2016 that my attention returned to it and I asked Walter for feedback. That resulted in a flurry of work to clean everything up and add some new features, with additional help from Gil and Jon.

Syntax

```
Call Showit .context~name , [output file] , item [ , item ...]
```

Note: The **output file** defaults to .context~name with "_help.txt" appended.

An **item** is one of the following, with an example for each:

- A whole file:
  - '1 * C:\Files\My_file'
  - '1 eof C:\Files\Your_File'
- A piece of a file:
  - ' 2 8 C:\Files\Another_File'
  - '20 * C:\Files\Another_File'
- A Classic stemname:

  mystem.

  Where the stem is numeric and mystem.0=the number of stems.

- An Object stemname:

  mystem.

  where mystem.0 is missing or not a whole number. The stemname will be sorted.

- A simple literal string:

  'Report header for XYZ Corp'

- A variable name and definition:

  - '&me='.context~name
  - '&me= Some prose'

  to be substituted in the output. If the argument starts with "&,then substitution is not done. This allows you to show the definition required for substitution. See the Help for an example.

- Options

  A string that starts with ':' triggers Options processing. Valid options to change the defaults are:

  - NOShow - Don't display the output file
  - Quiet - Skip the Exit message when NOSHOW is used
  - NOTepad - Allow testers to use NotePad

  Any number of Options arguments can be used and any option can be negated by starting it with 'NO' (or not) as needed.

Substitution

Variable definitions are saved in the order encountered as the arguments are processed. Everything from the & to the '=' becomes the varname. There are no restrictions on a varname, other than an *exact match* must be found for substitution to occur.

Each output line is examined for *every* variable, in the same order they were collected, and if found the substitution is made on the line using changestr() for all occurences.

This allows the clever programmer to have a variable contain another variable!

Uses

Its likely uses include:

- Extracting and displaying Help from code.
- Changing report headers and footers to another language.
- Collecting multiple raw data files into a single report, with headers and footers added.
- Tailoring email boiler-plate for the recipient.

  I do this with the THE editor for RexxLA Join/Renew and symposium Registration. That, however, preceeds this work.

Code

Here is the code: showit.rex.html

```
 /*
Date: 23 Oct 2015 16:01:05
Update: 31 Oct 2015 03:52:53 - Handle filespec and var substitution
Update: 11 Jul 2016 03:35:20 - Improve Help prose. Thanks to Walter
   Pachl
Update: 11 Jul 2016 22:02:50 - Fix for whole file and start to eof
Update: 12 Jul 2016 09:49:54 - Catch not enough args. Remove '
   interpret'
                               leftover from attempt at Regina
                                  compat.
Update: 12 Jul 2016 17:49:14 - Cleanup some dead code
Update: 14 Jul 2016 00:36:10 - Document double-quote "Do not
   substitute"
                               Display HELP subroutine as part of
                                  Help
Update: 14 Jul 2016 15:35:35 - Add options trigger ':' for NOSHOW,
   QUIET
Update: 15 Jul 2016 17:08:05 - Add NOQUIET to 'valids' Thanks Walter
   !
Update: 15 Jul 2016 23:33:07 - Allow NOSHOW and QUIET from cmdline
Update: 16 Jul 2016 00:25:19 - Add NOTEPAD option for cmdline
   testing
Update: 16 Jul 2016 06:54:02 - Show abbrev of options. Walter
Update: 16 Jul 2016 16:48:52 - Allow Help keywords from cmdline
Update: 17 Jul 2016 15:30:09 - Show that the Help keywords are
   optional
Update: 18 Jul 2016 14:46:26 - Use Seek in GetFile subroutine (
   Walter)
Update: 25 Jul 2016 16:34:34 - Comment code for testers (Rony
   Flatcher)
Update: 26 Jul 2016 14:00:12 - Use the modern lower case name
Update:  5 Aug 2016 06:08:54 - Use .context~name Fix example Add
   comments
Update: 12 Aug 2016 09:29:11 - Use single-quotes in first example
*/ beghelp=thisline()+1  /*
showit.rex:     See "Purpose", below
Copyright (C) 2015 Leslie L. Koehler
This is free software. See "Notice:" at the bottom of this file.

 Author: Les Koehler vmrexx@tampabay.rr.com

Purpose: Display the data items passed. Items can be a stem, a file
         (or a piece of one), a literal string or the definition of
         a variable that will be used for substituting its
            occurrences
         in lines from stems, literal strings, and/or files. For
            example:
              '&amp;varname=txt' or '&amp;varname='value
         As many items as needed can be used.
         A string that starts with ':' triggers Options processing.
         Valid options to change the defaults are:
           NOShow  - Do not display the output file
           Quiet   - Skip the Exit message when NOSHOW is used
           NOTepad - Allow testers to use NotePad
         Any number of Options lines can be used and any option can
            be
         negated by starting it with 'NO' (or not) as needed.
```

```
 Syntax: Call &amp;lcsme .context~name , [output fileid] , item [,
    item ...]

The output fileid defaults to the caller appended with '_help.txt'

Example: Call &amp;lcsme 'C:\MyRexxStuff\test.rex' , -- Caller
           , 'C:\Reports\Symposium.txt' ,        -- Output fle (
              optional)
           , 'Financial Data Report' ,          -- Literal string
           , stema. , stemb. ,                  -- Stems
           , '1 5 C:\Sigs\Lesk.txt' ,           -- Lines 1-5 of a
              file
           , '1 * C:\temp\junk.txt' ,           -- The whole file
           , '7 eof C:\temp\test.txt' ,         -- Line 7 to end-of-
              file
           , ':quiet noshow' ,                  -- Options
           , '&amp;lcme='lcme                       -- Variable
              definition

   Help: &amp;lcsme [&amp;helps]

Here is a real example, in the file &fullme:
*/
endhelp=thisline()-2
Call Parse_source
Parse Arg args
Call Parse_args args
If Arg()&lt;3 Then Call Exit 12 'At least 3 args are required'
--call dump
argarray = Arg(1,'a')
--say argarray~last 'items to process'
gothim?=0
gotfile?=0
outarray=.array~new           /* We'll collect the lines to write,
   here */
outfile=''
ampersands.0=0
--trace r
Do i = 1 To argArray~last
  Arg = argArray[i]
  Select
    When arg~isA(.stem) Then Do
--      Say 'Arg' i 'is a stem'
      Call handlestem(Arg)
    End
    When arg~isA(.string) Then Do
--      say 'Arg' i 'is a string:' Arg
      Call Handlestring Arg
    End
    When Arg=.nil Then Call Handledefaultfile
    Otherwise Say 'Arg' i 'is' Arg '(not handled)'
  End                                                        /*
     select */
End                                                            /*
   DO */
```

```
If outarray~last=.nil Then Do
  Call Exit 12 'There is no data to do variable substitution against.
     '
End
If ampersands.0&gt;0 Then Do                  /* Look for var
    substitution */
  Do l=1 To outarray~last
    line=outarray[l]
    If Pos('&amp;',line)&gt;0 Then Do             /* Need one here?
       */
      Do a=1 To ampersands.0                /* Yes. Walk down the list
          */
        Parse Var ampersands.a ampername'='val
        hit=Pos('&amp;',line)
        If hit&gt;1 Then Do                /* Room to test for double-
           quote? */
          If Substr(line,hit-1,1)='"' Then Iterate a    /* Skip if
             there */
        End
        If Pos(ampername,line)&gt;0 Then Do          /* Substitute if
            there */
          line=Changestr(ampername,line,val)
        End
      End
      outarray[l]=line                          /* Replace the line
         */
    End
  End
End
Call Sysfiledelete outfile
outobject = .stream~new(outfile)
--outObject~open("WRITE")
--If outobject~open('WRITE REPLACE') \= 'READY:'
If outobject~open('WRITE') \= 'READY:'
Then Call Exit 12 'Can''t open file:' outfile '('mystream~description
    ')'

outobject~arrayout(outarray,"LINES")
outobject~close()
--call dump
If show? Then Do
  Parse Upper Source os .
  If Left(os,7)='WINDOWS' Then Do
    uid=Word(Userid(),1)
    Select                          /* Give testers their favorite
       editot */
      When uid='Les' &amp; \notepad? Then ,
       'cmd.exe /c start /max c:\the\thec.exe "'outfile'"'
      When uid='Walter' &amp; \notepad? Then ,
       'cmd.exe /c start /max ked "'outfile'"'
      Otherwise                             /* Everyone else gets
         NotePad */
        'cmd.exe /c start /max notepad.exe "'outfile'"'
    End
  End
  Else 'cat "'outfile'" | more'                   /* Must be Linux
     type */
```

```
End
Else If \quiet? Then Call Exit 0 'created:' outfile
Call Exit
Exit

/* ============================================== */
HANDLESTEM: Procedure Expose outarray
  use Arg stem.
  If stem.0~datatype('w') Then Do i = 1 To stem.0
    outarray~append(stem.i)
  End
  Else Do ind over stem.~allIndexes
    outarray~append(stem.ind)
  End

  Return
/* ============================================== */
HANDLESTRING: Procedure Expose gothim? fullhim i outfile ,
 gotfile? ampersands. outarray (exposes) ucvalids abbrev ,
 keyword_parms? flags unknown? unknowns quiet? notepad? help?
--Say arg(1)
  If \gothim? &amp; i=1 Then Do
    fullhim=Arg(1)
    gothim?=1
  End
  If \gotfile? &amp; i=2 Then Do
    If Arg(1)\='' Then Do
      If Arg(1)\=fullhim Then Do
        outfile=Arg(1)
        gotfile?=1
      End
    End
  End
  If i&gt;2 Then Do
    Select
      When Datatype(Word(Arg(1),1),'W') Then Do
        Call Getfile Arg(1)
      End
      When Left(Arg(1),1)=':' Then Call Validate Substr(Arg(1),2)
      When Left(Arg(1),1)='&amp;' Then Call Next 'ampersands',Arg(1)
      Otherwise outarray~append(Arg(1))
    End
  End
  Return
HANDLEDEFAULTFILE: Procedure Expose fullhim i outfile gotfile?
  If \gotfile? &amp; i=2 Then Do
--    say 'Arg 2 is .nil setting the default:'
    outfile=fullhim||'_help.txt'
--    say '  'outfile
    gotfile?=1
  End
  Return
GETFILE: Procedure Expose (exposes) outarray i         /* Thanks to
   Jon Wolfers! */
  Parse Arg start last filename
  mystream = .stream~new(filename)
  If myStream~open('read') \= 'READY:'
```

```
    Then Call Exit 12 'Can''t open file:' filename '('mystream~
       description')'

  myStream~seek(start 'LINE') /* Position READ pointer to start point
      */
  myarray = myStream~arrayIn /* Read in the file, from 'start' to eof
      */
  If myStream~close \= 'READY:'
    Then Call Exit 12 'Can''t close file:' filename '('myStream~
       description')'

  If Datatype(last,'W') Then , /* Pick out our piece of MyArray */
   mysection = myArray~section(1, last+1 - start)
  Else ,
   mysection = myArray~section(1) /* The whole thing! */
  outarray = outArray~union(mysection)
Return
GETFILE: Procedure Expose outarray          /* Fails for multiple
    calls */
  Parse Arg start last filename
  outArray~appendAll(.stream~new(filename)~arrayIn~section(start,
     last+1 - start))
  Return

PARSE_SOURCE:
  Parse Source whatos how fullme
  Parse Value Reverse(fullme) With ext'.' em '\' mypath
  me=Translate(Reverse(em))':'
  sme=Substr(me,1,Length(me)-1)
  pad=Copies(' ',Length(sme))
  lcsme=Lower(sme)
  mypath=Reverse(mypath)'\'
  logfile=mypath||Lower(sme)'.log'
  ext=Translate(Reverse(ext))
  the?=ext='THE'
  rex?=\the?
  Parse version whatrexx rexxlevel rexx_release_date
  oorexx?=Pos('ooRexx',whatrexx)&gt;0
  regina?=Pos('REGINA',Translate(whatrexx))&gt;0
  If regina? Then Do
    Call Exit 99 'Sorry, Regina is not supported. ooRexx only.'
  End
  args=''
  opts=''
  If the? Then Do
    c='command'
    cn='command nomsg'
    m='macro'
  End
  Return
PARSE_ARGS:
  Call Init_vars
  If Words(args)=0 | how='COMMAND' Then Call Help
  Else Return
VALIDATE:
--trace r --call dump --trace r
  wds=Words(Arg(1))
```

```
  ucargs=Translate(Arg(1))
  Do w=1 To wds
    wd=Word(ucargs,w)
    ok?=0
    Do v=1 To Words(ucvalids)
      If Abbrev(Word(ucvalids,v),wd,Word(abbrev,v)) Then Do
        ok?=1
        Leave v
      End
    End
    If ok? Then Do
      ucwd=Word(ucvalids,v)
      z=Word(flags,v)                              /* Set flags
         indirectly */
      If z='notepad?' Then Do
        If Left(ucwd,2)\='NONO' Then Do
          Call Value z'.'w,1                        /* Set positional
             flag */
          Call Value z,1                                /* Set arg
             flag */
        End
        Else Do
          Call Value z'.'w,0                        /* Set positional
             flag */
          Call Value z,0                                /* Set arg
             flag */
        End
      End
      Else Do
        If Left(ucwd,2)\='NO' Then Do
          Call Value z'.'w,1                        /* Set positional
             flag */
          Call Value z,1                                /* Set arg
             flag */
        End
        Else Do
          Call Value z'.'w,0                        /* Set positional
             flag */
          Call Value z,0                                /* Set arg
             flag */
        End
      End
      argix.wd=ucwd
    End
    Else Do
      Call Value 'unknown?.'w,1
      unknown?=1
      unknowns=unknowns wd
    End
  End
--all dump
  If help? Then Call Help
  If unknown? &amp; keyword_parms? Then Do    /* Allow parms after
     keywords */
    kwdptrs=''
    kwds=''
```

```
    Do u=1 To wds                  /* Get the kwds in left to right
       order */
      wd=Word(ucargs,u)
      If \unknown?.u Then  Do                          /* Found a
         keyword */
        kwdptrs=kwdptrs u
        kwds=kwds wd
      End
    End
    kwdctr=Words(kwdptrs)
    Do p=1 To kwdctr                        /* Get the parms for each
       kwd */
      pix=Word(kwdptrs,p)                               /* Index into
         args */
      If pix+1&lt;wds &amp; p&lt;kwdctr Then Do              /*
         Another kwd later */
        piy=Word(kwdptrs,p+1)                           /* Ptr to next
           kwd */
        If piy\='' Then Do
          piy=piy-1                               /* Back up to prev
             word */
          pwords=piy-pix                /* Number of words between
             kwds */
        End
        Else Do
          Iterate
        End
      End
      Else Do                             /* TAILOR TO SUIT! Last
         keyword */
        If pix&lt;wds Then Do                          /* Something
           follows it */
--          If Word(ucargs,p)='FILE' Then pwords=wds-pix /* Get all
    of it */
          If Wordpos(Word(ucargs,p),keyword_parms)&gt;0 Then ,
           pwords=wds-pix                                /* Get all of
              it */
          Else pwords=1                                 /* Just one
             word */
        End
        Else pwords=0
      End
      Do u=pix+1 To pix+pwords                 /* Reset unknown?.
         flags */
        Call Value 'unknown?.'u,0         /* For parms that go with
           kwds */
      End
      vname=Word(kwds,p)                                /* Name of the
         var */
      vname=argix.vname
      vval=Subword(args,pix+1,pwords)                /* Value of the
         var */
      Call Value vname,vval                                    /* Set
         it */
    End
    unknowns=''                                             /*
       Reset */
```

```
    unknown?=0
    Do u=1 To wds          /* Accumulate any args that are still
       unknown */
      If unknown?.u Then unknowns=unknowns Word(args,u)
    End
  End
  unknown?=unknowns\=''
 --   Call dump
  If unknown? Then Call Exit 8 'Unknown option(s):' unknowns
  Return
INIT_VARS:
  valids='?  /? -? Help /Help -Help --Help'  /* Keywords
                  */
  abbrev='1  2  2  1    2     2     3     '  /* Minimum abbreviation
        */
  flags=Copies('Help? ',Words(valids))       /* Flag to set for
     keyword */
  helps=valids
  valids=valids 'SHOW  NOSHOW QUIET NOQUIET NOTEPAD' --&lt; your
     keywords
  abbrev=abbrev '1     3      1      3      3' --&lt; your
     abbreviations
  flags=flags   'show? show?  quiet? quiet? notepad?' --&lt; your
     flagnames
  flags=flags 'Unknown? Keyword_parms?'         /* Always the last
     ones */
  Do f=1 To Words(flags)
    v=Word(flags,f)
    Call Value v'.'f,0                    /* Initialize positional
       flag */
    Call Value v,0                             /* Initialize arg
       flag */
  End
  show?=1
  last=Words(helps)
  hhelp=''
  Do h=1 To last                        /* Build the Helps line
     variable */
    If h\=last Then hhelp=hhelp || Word(helps,h) '| '
    Else hhelp=hhelp||Word(helps,h)
  End
  helps=hhelp
  unknowns=''
  unknown?.=0
  ucvalids=Translate(valids)
--  msg.0=0
  keyword_parms?=0
--  keyword_parms='TO FILE PATH'
--  Parse Value '' With file path To
--  msg.0=0
  exposes='sme lcsme me msg. c cn m myrc pad quiet? notepad?' ,
   'help? mypath log? the? rex? logfile oorexx? regina? show? fullme'
      ,
   'beghelp endhelp helps'
  Return
MSG: Procedure Expose sme me rex? the?
  If rex? Then Say me Arg(1)
```

```
  Else 'msg' me Arg(1)
Return
EMSG: Procedure Expose sme me emsg rex? the?
  If rex? Then Say me Arg(1)
  Else 'emsg' me Arg(1)
Return
NEXT:
  Parse Arg !stem,!val
  If \Datatype(Value(!stem'.0'),'W') Then Call Value !stem'.0',0
  !ix=Value(!stem'.0')+1
  Call Value !stem'.0',!ix
  Call Value !stem'.'||!ix ,!val
  Return
THISLINE:
  Return sigl
DUMP:
  the?=0
  If the? Then Do
--  Interpret dumpvars('K')
--  Interpret dumpvars('K','zz')
    Interpret oodumpvars(,'zz')
    Interpret zz
  End
  Else Do
--  Interpret oodumpvars('K')
--  Interpret oodumpvars('K','zz')
    Interpret oodumpvars(,'zz')
    Interpret zz
  End
  Exit
VALUEOF:
  Arg !_!label
  Signal Value !_!label
  Return
HELP:
  xmpstart=thisline()-1
  xmpend=thisline()+9
  more=''
  if symbol('ARGS')='VAR' then more=':'args /* Allow options */
  outvar=beghelp endhelp fullme /* startline endline &amp; file to
     extract from */
  Call Showit fullme ,, outvar ,              /* Caller , data to
     show  */
   ,more ,       /* Allow options to be tested from a command window
      */
   ,"&amp;lcsme="lcsme , '&amp;sme='sme , '&amp;this file.='fullme ,
      /* Substitutions */
   ,"&amp;helps="helps , '&amp;fullme='fullme , xmpstart xmpend
      fullme
-- Note that the above 2 lines are NOT substituted when &amp;sme
   displays them.
-- That is because of the double-quote before the first '&amp;'.
  Call Exit
EXIT: Procedure Expose sme me sigl msg emsg rex? the?
  Parse Arg myrc mymsg
  mysigl=sigl
  If myrc='' Then myrc=0
```

```
  If myrc\=0 &amp;amp; mymsg\='' Then Do
    Call Emsg mymsg
    Call Msg 'Enter' sme 'HELP for help'
    Call Emsg 'Rc='myrc
  End
  Else If myrc=0 &amp;amp; mymsg\='' Then Call Msg mymsg
  If myrc\=0 Then Call Msg 'Exit called from line' mysigl
  Exit myrc
LOGIT: Procedure Expose (exposes) sigl
--trace r
  mysigl=sigl
  Parse Arg logargs
  If logargs='' Then logargs=Sourceline(mysigl+1)
  Parse Value Right(Space(Date(),0),9,0) Time('L') With ds ts
  logline=ds ts logargs
  If Arg(2,'E') &amp;amp; Arg(2)\='' Then Do
    Parse Value Arg(2) With his_sigl him
    logline=logline '&amp;gt;' him '@' his_sigl
  End
  Else logline=logline '@' mysigl
logfile=mypath||Lower(sme)'.log'
If oorexx? Then Do
  .stream~new(logfile)~~lineout(logline)~close
--  writeline='.stream~new("'logfile'")~~lineout("'logline'")~close'
--  Interpret writeline /* Bypass Regina prescan */
End
Else Do
  Call Stream logfile, 'C', 'OPEN WRITE APPEND'
  Call Lineout logfile,logline
  Call Stream logfile,'C', 'CLOSE'
End
Return
/* --- End of skeleton code --- Put subroutines below: */

/* Notice:
    This program is free software: you can redistribute it and/or
       modify
    it under the terms of the EPL (Eclipse Public License) as
       published by
    the Open Source Initiative, either version 1.0 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    EPL for more details.

    You should have received a copy of the EPL along with this
       program.
     If not, see:
    http://www.opensource.org/licenses/eclipse-1.0.php
*/
```