2019 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 2019 edition is registered under ISBN 000–0-0-0-0000-0-0-0

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 2019 symposium was held in Hursley, United Kingdom from 22 Sep 2019 to 25 Sep 2019.

[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

Tutorial: From Rexx to ooRexx – Rony G. Flatscher

Date and Time

23 Sep 2019, 19:00:00 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 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"

2019 International Rexx Symposium

Hursley, Great Britain

(September 2019)

© 2019 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 90'ies
  - OO-version of Rexx presented to the IBM user group "SHARE"
  - Developed since the beginning of the 90'ies
    - Originally led by IBM's Simon Nash (UK, Hursley)
    - Later led by IBM's Rick McGuire (USA)
  - 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 completely rewritten
    - 32-bit *and* 64-bit versions possible for the first time
    - New OO-APIs into the ooRexx kernel
      - e.g. BSF4ooRexx allows for implementing Java methods in Rexx !
- Latest release as of September 2019
  - ooRexx 4.2, Feb 24, 2014
  - AIX, Linux, MacOS, Windows
- ooRexx 5.0 in beta, about to be released?

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,), MacOS, Windows
    - Consolidated (great!) PDF-rendered documentation!

New Procedural Features, 1

- 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 (ADT)
  - "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 ADTs
  - 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*

About Implementing ADTs, 1

- Rexx and ADTs
  - Cannot define routines confined to a datatype!
  - Attributes can be encoded as
    - Rexx strings, e.g.

```
birthday="19590520 13:01"
```

    - Rexx stems, e.g.

```
birthday.date="19590520"
Birthday.time="13:01"
```

  - Quite complicated and can be error prone
    - Rexx programmers must know exactly the structure and all operations to implement!

About Implementing ADTs, 2

- ooRexx
  - Designed to easily implement ADTs
  - Directives

```
::CLASS adt_name
::ATTRIBUTE attr_name
::METHOD meth_name
```

  - An implemented ADT is sometimes termed "class", sometimes "type", sometimes "structure"
  - "Black box"
    - Rexx users do not need to know any implementation details in order to use classes/types/structures !

About Objects and Messages

- "object"
  - A synonym for "value of a specific type", "instance"
  - Possesses all attributes and methods of its class
  - Only reacts upon receiving messages
    - Message operator ~ (tilde, dubbed "twiddle")
    - Followed by a *message name*, optionally with arguments in parenthesis
    - Searches and invokes the method with the same name as the message name and returns any return value from the method

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 its name                */
say "My name is:" myDog~name    /* now ask the dog for its name */
myDog~bark               /* come on show them who you are!       */

::class  Dog             /* name of the implemented ADT          */
::attribute name         /* 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 its name                */
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"               */
::attribute name         /* 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 like big dogs
                            do, all in uppercase! :)  */
  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: *http://www.facultas.at/flatscher*
- TBD*: http://www.RonyRexx.net*

2

40 Years of Rexx - a personal view – Mike Cowlishaw

Date and Time

24 Sep 2019, 10:00:00 CET

Presenter

Mike Cowlishaw

Presenter Details

Mike Cowlishaw is the creator of Rexx and has worked in both hardware and software design and is currently the Editor of the IEEE 754 Standard for Floating-Point Arithmetic. He has long been interested in the human aspects of computing, including the Rexx and Java programming languages, colour perception, neural networks, text editing, mapping, panorama viewers, and decimal arithmetic. He is an IBM Fellow (retired), a Fellow of the Royal Academy of Engineering, and a Visiting Professor in the Department of Computer Science at the University of Warwick, UK.

Session Abstract

Why does the name 'Rexx' have a double-x? When and how did Rexx get started? What was the context? How did Rexx get added to IBM operating systems? And why is its decimal arithmetic so important? In this talk, Mike Cowlishaw, author of the Rexx language, will answer these and other questions. He'll also share his own thoughts on the background, design, and highlights (and a couple of lowlights) of the evolution of Rexx.

40 years of Rexx — a personal view

Hursley

23 September 2019

Mike Cowlishaw
http://speleotrove.com/mfc/

Rexx40

Overview

40 years of Rexx; a timeline …

Overview

40 years of Rexx; a timeline …

… or, in reality, 50 years …

Rexx roots go back 50 years…

In 1969 my mathematics teacher at Monkton Combe School, Julian Bewick, taught me to program using a pseudo-assembler (paper-executed) called '*minlan*'; I soon learned FORTRAN IV and wrote an interpreter for *minlan* …

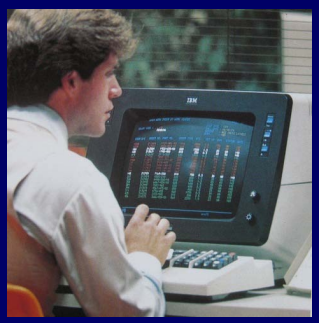
… and on 28 September 2019 we'll be celebrating those early days at the school!

Then … (1970-1974)

- MFC: Pre-University student with IBM (1970)
  - PL/I compiler, compiler for *minlan*, *etc.*
- BSc Electrical & Electronic Engineering (1974)
  - University of Birmingham
  - Vacation jobs at IBM Hursley and IBM Bristol
  - lots of caving
  - First paper: "The Characteristics and Use of Lead-Acid Cap Lamps" (1974)

Then … (1974-1979)

- IBM Hursley: Test Tools Team (1974–1979)
  - building hardware for testing terminals such as the 3279 …

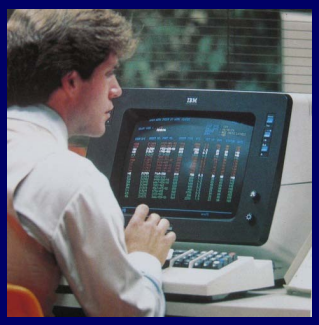


The team (December 1978)

(Ron Bowater, Doug Buttimer, Dave Milward, MFC)

Then … (1974-1979)

- IBM Hursley: Test Tools Team (1974–1979)
  - building hardware for testing terminals such as the 3279 …



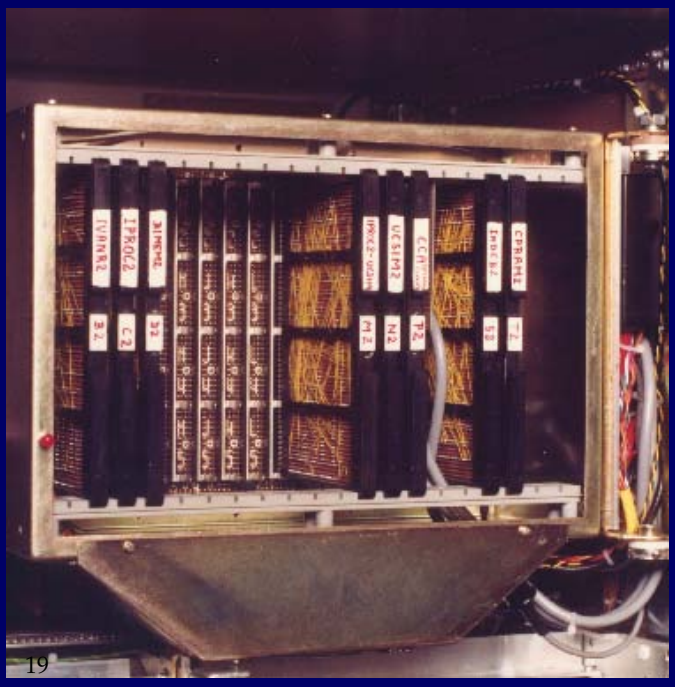
The team (October 2018)

(Dave Milward, Doug Buttimer, Ron Bowater, MFC, Nick Butler)

… 2019 reunion today!

Microlink

- Used existing coax terminal link (ANR) to attach bipolar microcomputers (such as the 250ns Signetics 8X300) to mainframe
- Software included OS, Compilers, circuit layout …



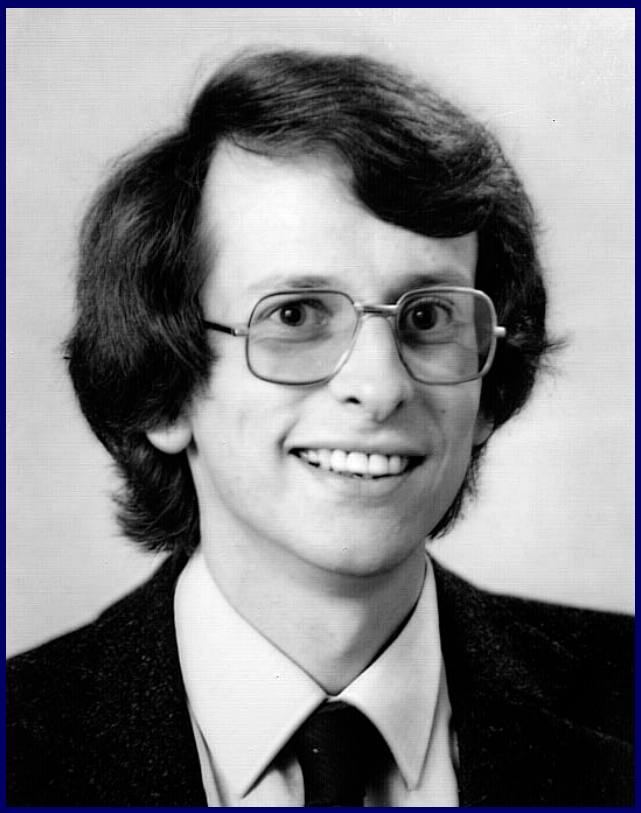
Own-time projects

- Mostly PL/I and S/360 Assembler
  - Archaeological mapping (1974)
  - Cave surveying programs (1976)
  - Several compilers and interpreters (1976+)
- STET, a STructured Editing Tool (1977)
  - and *lots* of other VM/CMS tools
- Rex (started 20 March 1979)
  - a biggie: 4,000 hours to 1982

How old was I?

well, 40 years is 40 years …

How old was I? (1979 pix)



After Cueva Toyu — Guinness ceremony

Why Rex?

- CMS had EXEC … a bit like DOS BAT

```
&CONTROL OFF
&IF &INDEX EQ 0 &GOTO -GO
EXEC DCOPT DROP
&IF &RETCODE GE 12 &EXIT
-GO
&STACK RT …
```

- EXEC 2: clean design, but just as ugly
  - language committee (Stephenson *et al*.)
  - hooks for vanilla CMS by Michel Hack

The first Rex programs

- ADDR EXEC … searches nickname file for nickname, displays name and address
- SEND EXEC … send file to a local user
- CONC XEDIT … concatenate & flow macro
- … and lots of testcases

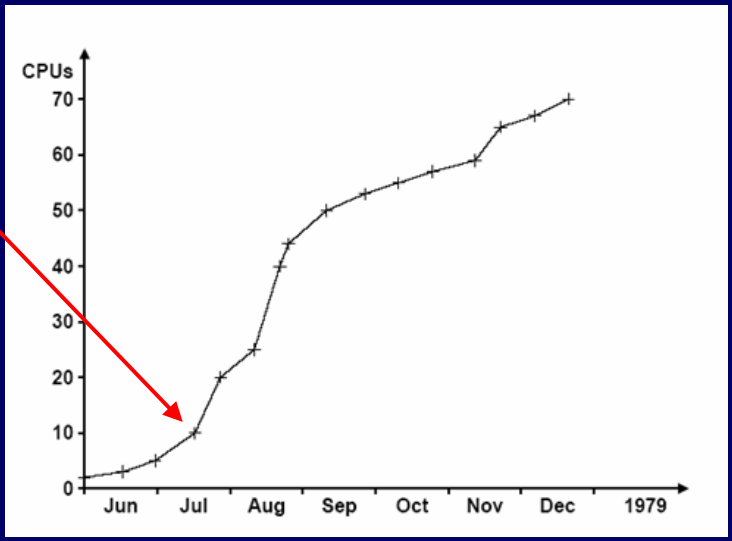
Who used Rex?

- First distributable code was in May 1979; until then, only the one user
- The first real users (pioneers, guinea pigs, trend-setters, …) were
  - Ray Mansell (Hursley, UK)
  - Les Koehler (Raleigh, NC)

  … lots of useful feedback

How did it catch on?

- Internal IBM network, VNET, rapidly growing
- VM Newsletter (Peter Capek)
- Word of mouth, Xmas card …
- Add-ons (Steve Davies' functions and many others)



15

Was there a Rex motto?

- Sort of. Pinned to the wall over my desk in Hursley was …

23

16

Was there a Rex motto?

- Sort of. Pinned to the wall over my desk in Hursley was …

> Keep the language small

….. < 32 KB!

Why decimal arithmetic?

- One type = characters = decimal
  - avoids many problems (*e.g.*, 0.9/10 = 0.089999996)
  - see http://speleotrove.com/decimal/decifaq1.html
- Current is third iteration, May—July 1981
  - lots of input from language committee
  - … and from users in 43 countries
  - … *and* from a noisy *'no more changes'* lobby

Why are '!' and '?' in symbols?

- I always intended to complete the arithmetic by adding Infinity and NaN

  ! was to be used for Infinity
  ? was to be used for NaN

- Code freeze for product meant these and other changes (*e.g.,* stream I/O) omitted
  - so EXECIO had to be used for files

1981: How did it become 'official'?

- Internal CMS included XEDIT-editor-based tools, almost all developed using Rex

- Claude Hans in Endicott decided to add Rex even before EXEC 2 shipped; Rick McGuire involved from March 1981

- SHARE talk in 1981 … Ted Johnston (SLAC) asked IBM CEO (Frank Cary) for Rex

1981: A setback…

- IBM PC announced in August

  The very first thread on the new 'IBMPC FORUM' was: "who's writing Rex for the PC?" − many keen volunteers …

- … but a group in San Jose was officially funded to write Rex for PC − so no one else tried; unfortunately they wrote it in Pascal − so the project failed

1982: Why 'REXX'?

- Trademark search in 1982 found an unrelated product called **`Rex-80`**

- Lawyers insisted that 'X' be added …
  … and 'REXX' not be used in product name
  … and acronym was expanded

- Estimated cost of the change: $1,000,000+