Communications and Event Handling with REXX

Rainer F. Hauser

May 1992

Some Questions

REXX is a sequential procedure (macro, control or glue) language. Is it really, or could it be that it is actually a programming language? What about REXX and concurrency?

Communications:

Is REXX the right choice for programming communications software? Does it provide the necessary constructs for such programs? What about the performance?

Event Handling:

Is REXX suitable for general event handling? What is missing today for writing such programs? What are the events which fit the paradigm of REXX?

Three REXX Extension Packages:

A kind of answer ”by doing” to some of these questions:

- REXXIUCV: REXX Interface to IUCV
- REXXSOCK: REXX Interface to TCP/IP Socket Calls
- REXXWAIT: REXX General Purpose Event Handling with a Central Wait Function

REXX and Concurrency Today

A REXX program can process events sequentially. To do so, it needs the possibility to find out when an event has occurred, but has not yet been consumed.

The following REXX statements determine whether a console event is pending:

```
if externals()>0 then ...
```

When no event is currently pending, it needs the possibility to wait for an event.

The following REXX statements wait for a console event:

```
say 'Enter your name, please.'
parse external name
```

To avoid being blocked despite a pending event which could be processed, it needs the possibility to wait for one event within a given list of events.

Today, a REXX program can

- sometimes not determine whether a specific event is pending
- often not wait for a specific event
- not wait for one event within a given list of events

REXX and Communications

One system facility and the two REXX extension packages REXXIUCV and REXXSOCK provide communications in REXX:

APPC:

APPC is available via the SAA Common Programming Interface Communications (CPI–C) and the Callable Service Library (CSL).

IUCV:

IUCV is a communications facility available on VM systems. The REXXIUCV program provides access to it from REXX on VM/CMS. Therefore, it is a system–dependent communications extension for REXX.

TCP/IP:

TCP/IP is a communications facility available on various different platforms. The REXXSOCK program provides access to it (i.e. to the socket calls) from REXX on VM/CMS. Therefore, it has been designed as a system–independent communications extension for REXX.

REXXIUCV:

Syntax: `result = IUCV(subfunction, arg1, …, argn)`

Subfunctions:

- INIT, TERM, QUERY, WAIT, ...
- CONNECT, ACCEPT, SEVER, ...
- SEND, RECEIVE, ...

Examples:

```
tempdata = IUCV('CONNECT','RFH',255,'No')
parse var tempdata pathid msglim .
inttype = IUCV('WAIT',600,'NOWAIT')
nextint = IUCV('QUERY','NEXT')
parse var nextbuf . buftype bufpathid rest
...
```

Problems:

- Assembler Paradigm vs. REXX Paradigm
- Special Purpose Wait Subfunction

REXXSOCK:

Syntax: `result = TCPIP(subfunction,` $\texttt{arg}_1$`, …,` $\texttt{arg}_n$`)`

Subfunctions:

- INIT, TERM, QUERY, GETHOSTID, ...
- SOCKET, BIND, CONNECT, ACCEPT, CLOSE, ...
- WRITE, READ, SEND, RECV, ...

Examples:

```
inetaddr = 'AF_INET 1291 9.4.3.2'
socketid = TCPIP('SOCKET')
tempdata = TCPIP('CONNECT',socketid,inetaddr)
...
```

Problems:

- C Paradigm vs. REXX Paradigm
- Functions such as CONNECT and READ block the caller
- Data can be encoded as ASCII or EBCDIC

Common Design Decisions

Both packages are based on the following design decisions:

- The status of the communications facility is kept by the REXX extension package and can be determined by the REXX program.
- The status of the communications facility should not be destroyed when the REXX program terminates.
- Individual IUCV primitives or TCP/IP socket calls should be provided as individual function calls to REXX. In other words, there should be a one–to–one mapping between REXX functions and IUCV primitives or TCP/IP socket calls, respectively.
- A REXX program should be allowed to process events selectively as appropriate to the program (and the programmer).
- Return codes are presented to the REXX program in the REXX variable RC.
- Limits such as the maximum length of messages are necessary, but should be easy to change. (Such limits should also make sense to human beings and not to computers. Therefore, values such as 1000 are a better choice than values such as 1024).

Experiences

REXX as a programming language is well suited for communications software, but with the current language features, there are some limitations and inconveniencies:

Conversions:

REXX does not provide functions to convert ASCII strings to EBCDIC strings and vice versa.

```
astring = TCPIP('READ',socketid)
estring = A2E(astring)
...
```

Event Handling:

REXX does not provide functions to wait for one of several expected events. Assume that a REXX program needs to wait for either an IUCV or a TCP/IP event.

```
event = WAIT('IUCV PATH 5','TIME 10MIN')
event = WAIT('TCP/IP READ 4','TIME 10MIN')
event = WAIT('IUCV PATH 5','TCP/IP READ 4')
...
```

REXX and Event Handling

The REXX extension package REXXWAIT on VM/CMS provides basic and advanced event handling in REXX through a central wait function for REXX programs and a low–level interface for REXX extension programs.

State of the Art (The REXX Handbook):

- Amiga REXX: IPC (waiting on message port)
- REXX for Tandem: DELAY function and TACLIO IPC
- REXX for Unix: Plan for IPC (SOCKETS, STREAMS)
- REXXIUCV: IUCV('WAIT',seconds)
- ...

Common Events:

- Keyboard and Mouse: Character Oriented, Block Mode, Window Applications ...
- Time: Relative and Absolute Time, Time Events in Files ...
- Mail: Messages, Notes ...
- Synchronization: Inter–Process Communication, Locks, Semaphores ...

REXXWAIT

WAIT Function Syntax:

```
event = WAIT(event₁ args₁, …, eventₙ argsₙ)
```

Events:

- Basic Events: CONS, WNG, MSG, ..., MAIL, FILE, TIME
- Additional Events: IUCV, TCP/IP

Examples:

```
event = WAIT('CONS NOREAD','TIME 10MIN')
event = WAIT('CONS','MSG','FILE MY TIMEFILE A6')
event = WAIT('TIME ==:=0:00','ALL')
event = WAIT('IUCV TYPE 3 PATH 15')
event = WAIT('TCP/IP READ 15 WRITE 20 21','CONS')
event = WAIT('TIME 10S','TIME 10:30:15','TIME')
...
```

SETVALUE Function Syntax:

```
result = SETVALUE(event args)
```

QUERYVALUE Function Syntax:

```
result = QUERYVALUE(event args)
```

REXXWAIT REXXTRY Sample Session

Sample session with REXXTRY, the facility to interactively execute REXX instructions:

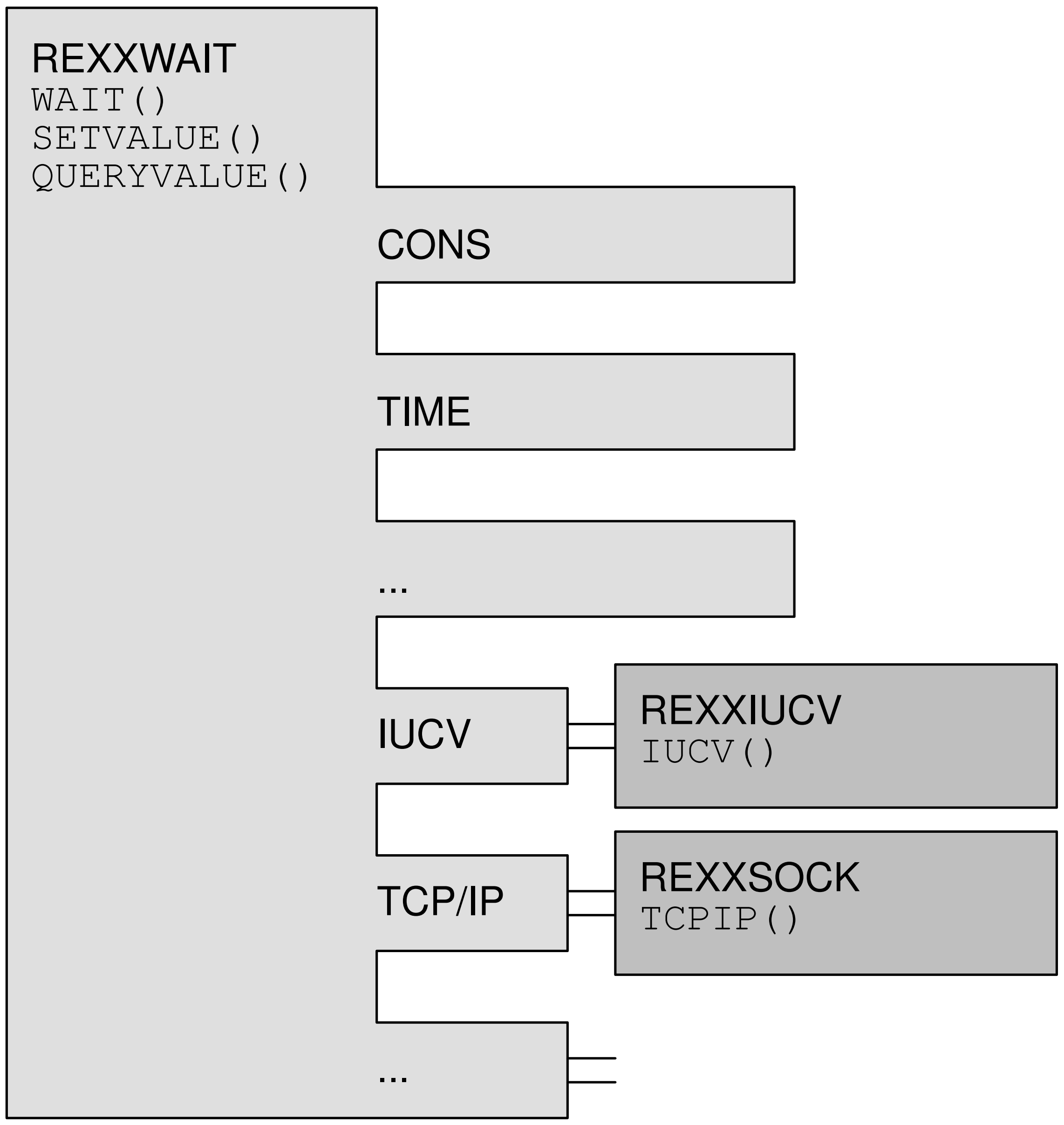
```
say setvalue('MAIL CLASS * NOHOLD')
OFF
R; <REXXTRY> ..............................................
say wait('MAIL','CONS')
RDR FILE 0050 SENT FROM NET    PUN WAS 1991 RECS 0022 ...
File (2404) spooled to HAUSER -- origin ZURLVM1(RFH) ...
MAIL 0050
R; <REXXTRY> ..............................................
say queryval('MAIL 50')
1992/04/16 05:39:41 ZURLVM1(RFH) NOHOLD A 0 0 PROFILE EXEC
R; <REXXTRY> ..............................................
say queryval('MAIL 50 TAG')
2 NetData
R; <REXXTRY> ..............................................
say queryval('MAIL 50 NETDATA')
2 ZURLVM1(RFH) ALMVMD(HAUSER) 19920416133936499588 Ack
R; <REXXTRY> ..............................................
say queryval('MAIL 50 NETDATA 1')
Note
R; <REXXTRY> ..............................................
say queryval('MAIL 50 NETDATA 2')
File A2.PROFILE.EXEC
R; <REXXTRY> ..............................................
```

REXXWAIT REXX Sample Program

Sample program using REXXWAIT and REXXSOCK (without the necessary error testing):

```
...
address command 'RXSOCKFN LOAD'
call TCPIP 'INITIALIZE', 'TCPIP'
if rc<>0 then exit rc
s = TCPIP('SOCKET')
call TCPIP 'BIND', s, 'AF_INET 1952 9.4.3.2'
call TCPIP 'LISTEN', s, 5
call SETVALUE 'TCP/IP SOCKET' s 'NON-BLOCKING'
do forever
  status = TCPIP('QUERY', 'STATUS')
  parse var status init iucvstate reason
  if iucvstate<>'Connected' then exit 3000
  eventd = WAIT('TCP/IP READ' s, 'CONS', 'TIME 1H')
  parse upper var eventd handler rest
  select
    when handler='TCP/IP' then do
      desc = TCPIP('ACCEPT', s)
      parse var desc d caf cport cipaddr
      ...
      call TCPIP 'CLOSE', d
    end
    when handler='CONS' then leave
    otherwise nop
  end
end
call TCPIP 'TERMINATE'
address command 'NUCXDROP RXSOCKFN'
...
```

REXXWAIT Low–Level Interface



Through the low–level interface provided by REXXWAIT, other programs (such as REXXIUCV and REXXSOCK) can export an event name (such as IUCV and TCP/IP) and some branch addresses for communicating with the REXX programs using the functions provided by REXXWAIT.

REXXWAIT Assembler Sample

The following /370 Assembler code shows the code to register an event handler:

```
         LA     R1,RXWPLIST          R1 -> PLIST
         SVC    202                  Call CMS
         DC     AL4(1)               Error
or
         CMSCALL PLIST=RXWPLIST,CALLTYP=PROGRAM,...
...
         DS     0F                   Alignment
RXWPLIST DC     CL8'RXWAITFN'        CMS command
RXWCMD   DC     CL4'SET'             RXW command
         DC     XL4'00000000'        Special fence
RXWNAME  DC     CL8'TCP/IP'          Registered name
RXWWTECB DC     F'-1'                Address of ECB
RXWWAIT  DC     F'-1'                BAL address WAIT
RXWWAITE DC     F'-1'                BAL address WAIT-E
RXWSETV  DC     F'-1'                BAL address SETV
RXWQRYV  DC     F'-1'                BAL address QRYV
         DC     F'-1'                Not used yet
         DC     F'-1'                Not used yet
         DC     F'-1'                Not used yet
...
```

The program registers the event handler name 'TCP/IP' and provides the following BAL or ECB addresses (or F'–1'):

- For WAIT: Wait ECB, WAIT and WAIT–E
- For SETVALUE: SETV
- For QUERYVALUE: QRYV

REXXWAIT Register Conventions

When REXXWAIT passes the control to an event handler, the following registers are used:

```
R0   length of arguments
R1   pointer to arguments
R2   call sequence flag (for WAIT only)
R12  address of exit routine (base register)
R13  save area for general purpose registers
R14  return address
R15  same as R12
```

The event handler passes the following data in the registers back to REXXWAIT:

```
R0   length of result string
R1   pointer to result string
R15  return code (0, 1 or error return code)
```

Example:

Event Handler ’ABCD’:

```
WAIT(’ABCD This is the argument string’)
  == ’ABCD This is the result string’
```

The event handler sees the arguments ’This is the argument string’ and returns the result ’This is the result string’ to the REXXWAIT program which subtracts or adds the event handler name ’ABCD’, respectively.

Considerations

Portability:

Some of the basic events are generally available on all operating system platforms on which REXX is implemented. Other events require different arguments. Again others may not be available at all.

```
WAIT('TIME 10:30:15')
WAIT('TIME 2HOURS 15MINUTES')
WAIT('CONS READ')
WAIT('FILE MY TIMEFILE A6')
WAIT('FILE C:\REXXSYS\MY.TIM')
WAIT('FILE' mytimefile)
SETVALUE('MSG ON')
SETVALUE('SMSG VMCF')
```

Operating System Support vs. REXX Support:

The same functions can be provided to a REXX program either as Operating System facilities or as REXX built–in functions:

- EXECIO vs. linein() and lineout() etc.
- WAKEUP vs. wait() etc.

The WAKEUP program (version 5.5) has strongly influenced the design and implementation of REXXWAIT!