RXIUCVFN
VM REXX IUCV Function Package

Reference Manual

Document Number RXIUCVFN/1.36

June 16, 1993

Rainer F. Hauser

IBM Research Laboratory
Zurich, Switzerland
RFH at ZURLVM1

IBM Internal Use Only

Contents

Figures

1.0 Introduction

RXIUCVFN (also called REXXIUCV) is an external REXX function package that, when loaded as a nucleus extension, provides a comprehensive set of functions to make use of the IUCV facility (either native or using the CMS IUCV interface). Applications using IUCV usually have to be programmed in assembler language. Thus, this package enables a programmer to implement and test draft or final solutions for procedures using IUCV within REXX. With this package a user may also gain some experience with IUCV. RXIUCVFN contains functions to handle the IUCV environment, IUCV paths and IUCV messages together with some auxiliary functions.

For those not familiar with IUCV (Inter-User Communications Vehicle), it is a communications facility that enables a program running in a virtual machine to communicate with other virtual machines (if such connections are allowed in the CP directory), with a CP system service such as *MSG (see the REXX-EXEC 'QUSER' below), and with itself (see the sample session below).

The user of the RXIUCVFN package is assumed to have at least some rough understanding of IUCV as described in the System Programmer's Guide (VM/SP System Programmer's Guide, SC19-6203), or more recent IBM manuals.

The RXIUCVFN package is available from the VMTOOLS conferencing disk as REXXIUCV package. It consists of a module, a CMS help file and this documentation. In addition, some examples are included.

Before a user can use any of the IUCV functions described below, he will need to enter a command that will cause these functions to be installed as a nucleus extension.

To install the IUCV functions, enter:

```
'RXIUCVFN LOAD'
```

To unload (de-install) the functions, enter:

```
'NUCXDROP RXIUCVFN'
```

To find out whether the functions have already been installed, enter:

```
'RXIUCVFN TEST'
```

The return code tells that the package has already been loaded (RC=0) or that it has not yet been loaded (RC=1).

Note: RXIUCVFN is available under the name "VM REXX Programming Support for IUCV" (REXXIUCV) and the Program Number 5785-LAT in several countries. (See the Availability Notice, GB11-8432, and/or the Program Description and Operations Manual, SB11-8433, for more information.)

2.0 Function Syntax

The general behavior of the REXX function IUCV is described in the following. Details about the behavior of the various subfunctions are presented in the next chapters.

2.1 The REXX Function IUCV

The package contains only one REXX function, called IUCV, and the name 'function' is used in this documentation to denote its subfunctions. The general syntax of all functions is

> **IUCV(function,arg1,arg2,...,argn)**
>
> Result: result-string

and all functions return a result string (with the different items separated by a blank) that can be parsed by REXX in the usual way. The function QUERY consists of some subfunctions specified in arg1. When they are referenced, names such as QUERY-VERSION are used to distinguish these subfunctions. In addition to the result string, the REXX variable 'RC' is set because the IUCV return code carries important information. (The target side of an IUCV path only knows whether the source side has purged a message when he tries to receive it and analyzes the return code.) As a rule, the result string is empty when the return code is not zero, but there is an exception: Receiving only a part of the message returns this part in the result string and sets a return code 5 according to IUCV to indicate that the receive buffer was too short.

The input parameters for the individual functions and the items in the result string together with their range and default values are described in alphabetical order in the following list:

cmsiucv Use of CMSIUCV interface ('Application'|'Cms'|'Native')
Default: 'NATIVE'
Input to INIT

cmsname CMSIUCV name (arbitrary characters up to 8 bytes)
Default: 'RXIUCVFN' (entered value padded with blanks)
Input to INIT
Output of QUERY-CMSNAME

data Message or reply data (arbitrary characters up to 100,000 bytes)
Input to SEND, REPLY
Output of RECEIVE, QUERY-REPLY

date Date of the package
Output of QUERY-VERSION

error Initialization error indication ('Yes'|'No')
Output of QUERY-STATUS

extint External interrupt type (hexstring 4 bytes)
Output of WAIT

how Wait keyword or number ('WAIT'|'NOWAIT'|arbitrary integer)
Default: 'WAIT'
Input to WAIT

init Initialized indication ('Yes'|'CMS'|'No')
Output of QUERY-STATUS

intstatus IUCV interrupt enabled indication ('On'|'Off')
Output of QUERY-STATUS

msgcls Source or target message class (see srccls and trgcls)
Input to QUERY-NEXT

msgid Message identification (integer)
Input to PURGE
Output of SEND, REPLY, QUERY-NEXT

msglength Maximal length of a message (integer between 0 and 100,000)
Default: whole length of the message
Input to RECEIVE
Output of QUERY-NEXT

msglim Maximal number of outstanding messages (integer between 1 and 10,000)
Input to CONNECT, ACCEPT
Output of CONNECT, ACCEPT, QUERY-NEXT

mvmid Monopolized vmid (arbitrary vmid up to 8 bytes)
Input to INIT
Output of QUERY-MVMID

name Name of the package ('RXIUCVFN')
Output of QUERY-VERSION

numintb Maximal number of interrupt buffers (integer between 1 and 1,000,000)
Default: 500
Input to INIT

numpath Maximal number of paths (integer between 1 and 2,000)
Input to INIT
Output of INIT

pathid Path identification (non-negative integer)
Input to ACCEPT, SEVER, QUIESCE, SEVER, SEND, PURGE, RECEIVE, REJECT, REPLY, QUERY-PATH, QUERY-NEXT, QUERY-REPLY
Output of CONNECT, ACCEPT, QUERY-PATH, QUERY-NEXT

pathlist List of pathids (see pathid)
Output of TERM, QUERY-PATH

pending Number of pending interrupt buffers (non-negative integer)
Output of QUERY-NEXT

prio Priority messages keyword ('PRIORITY'|'NONPRIORITY')
Default: 'NONPRIORITY'
Input to CONNECT, ACCEPT, SEND, REPLY
Output of QUERY-NEXT

prm Message or reply data in parameter list keyword ('PRMDATA'|'NOPRMDATA')
Default: 'NOPRMDATA'
Input to SEND, REPLY

qkeyword Query keyword
('STATUS'|'VERSION'|'CMSNAME'|'MVMID'|'PATH'|'NEXT'|'REPLY')
Input to QUERY

rpl Reply needed keyword ('REPLY'|'NOREPLY')
Default: 'NOREPLY'
Input to SEND
Output of QUERY-NEXT

rpllength Maximal length of a reply (integer between 0 and 100,000)
Input to SEND
Output of QUERY-NEXT

seconds Number of seconds to wait (integer between 1 and 3,600)
Input to WAIT

srccls Source message class (arbitrary integer)
Input to SEND, PURGE, QUERY-REPLY
Output of QUERY-NEXT

trgcls Target message class (arbitrary integer)
Input to SEND, RECEIVE, REJECT, REPLY
Output of QUERY-NEXT

type Interrupt buffer type (non-negative integer)
Input to QUERY-NEXT
Output of QUERY-NEXT

userdata User data (arbitrary characters up to 16 bytes)
Default: XL16'00' (entered value padded with X'00')
Input to CONNECT, ACCEPT, SEVER, QUIESCE, RESUME
Output of QUERY-NEXT

version Version of the package (currently 1.36)
Output of QUERY-VERSION

vmid VM userid or CP service (alphanumeric up to 8 bytes)
Input to CONNECT
Output of QUERY-PATH, QUERY-NEXT

Figure 1 summarizes the functions available in this package. The internals of the individual functions are described in some detail below. The functions are divided into four classes:

- functions to activate or deactivate IUCV
- functions to manipulate IUCV paths
- functions to manipulate IUCV messages
- auxiliary functions

```
(INIT,numpath,[cmsiucv,[cmsname],[mvmid]],[numintb])
(TERM)

(CONNECT,vmid,msglim,[prio],[userdata])
(ACCEPT,pathid,msglim,[prio],[userdata])
(SEVER,pathid,[userdata])
(QUIESCE,pathid,[userdata])
(RESUME,pathid,[userdata])

(SEND,pathid,data,srccls,trgcls,[prio],[prm],[rpl,[rpllength]])
(PURGE,pathid,srccls,msgid)
(RECEIVE,pathid,trgcls,[msglength])
(REJECT,pathid,trgcls)
(REPLY,pathid,data,trgcls,[prio],[prm])

(QUERY,'STATUS')
(QUERY,'CMSNAME')
(QUERY,'MVMID')
(QUERY,'VERSION')
(QUERY,'PATH',[pathid])
(QUERY,'NEXT',[type],[pathid],[msgcls])
(QUERY,'REPLY',pathid,[srccls])
(WAIT,seconds,[how])
```

Figure 1. Summary of functions in RXIUCVFN

Figure 2 on page 6 summarizes the result string returned by functions of this package. The different items in the result string can always be parsed and identified uniquely. Functions returning 'data' (message data or reply data) do not return any other items to guarantee this uniqueness. (This is one reason why the return code is set into the REXX variable 'RC' and is not delivered within the result string.) For QUERY-NEXT the items following 'type' are listed for all possible values of 'type'.

```
INIT              numpath
TERM              pathlist

CONNECT           pathid msglim
ACCEPT            pathid msglim
SEVER
QUIESCE
RESUME

SEND              msgid
PURGE
RECEIVE           data
REJECT
REPLY             msgid

QUERY-STATUS      init error intstatus
QUERY-CMSNAME     cmsname
QUERY-MVMID       mvmid
QUERY-VERSION     name version date
QUERY-PATH        pathlist | [pathid vmid]
QUERY-NEXT        pending [type pathid others]
                  [1 pathid msglim vmid prio userdata]
                  [2 pathid msglim userdata]
                  [3 pathid userdata]
                  [4 pathid userdata]
                  [5 pathid userdata]
                  [6 pathid msgid srccls[rpl rpllength]]
                  [7 pathid msgid srccls[rpl rpllength]]
                  [8 pathid msgid trgcls msglength[rpl rpllength]]
                  [9 pathid msgid trgcls msglength[rpl rpllength]]
QUERY-REPLY       data
WAIT              extint
```

Figure 2. Result string returned by functions in RXIUCVFN

2.2 Return Codes of the Functions

It is not usual for REXX built-in functions to return any return codes in the REXX variable 'RC'. But here the decision was made to make the IUCV (and other) return codes available to the user in the REXX variable 'RC'. (Note that not all possible IUCV return codes can occur because some are prevented by RXIUCVFN.)

Return codes set according to IUCV:

0 Normal return
1 Invalid path ID specified
2 Path quiesced - no sends allowed
3 Message limit exceeded
4 Priority messages not allowed on this path
5 Receive or answer buffer too short to contain message
6 Storage protection exception on send or answer buffer
7 Addressing exception on send or answer buffer
8 Message found but message class or path id invalid
9 Message has been purged
10 Message length is negative
11 Target communicator is not logged on
12 Target communicator has not invoked DCLBFR

13 Maximum path number for source communicator
14 Maximum path number for target communicator
15 No authorization found
16 Invalid CP system service name
17 Invalid function code in IPFCNCD
18 Value in IPMSGLIM exceeds 255
19 A previously declared buffer is still in use
20 Originator has severed this path
21 Parameter list data not allowed on this path
22 Send or answer buffer list invalid
23 Negative length in buffer list
24 Incorrect total length of buffer list lengths
25 PRMMSG option invalid with BUFLIST/ANSLIST option
26 Buffer list not on a doubleword boundary
27 Answer list not on a doubleword boundary
29 Any other return code from IUCV

Other return codes set by RXIUCVFN functions:

30 Invalid arguments
31 Not enough free storage available
32 No more path allowed
33 Message too long
34 Path neither connected nor pending
35 No corresponding interrupt buffer pending
36 Bad condition code from IUCV or return code from CMSIUCV
37 No initialization has yet been performed
38 Initialization has already been performed
39 The IUCV error bit was set
40 CMSIUCV name already in use by another program

2.3 General Remarks

This section presents some general remarks important for users of this package:

1. IUCV can either be used in synchronous mode (send-receive-reply sequences) or in asynchronous mode (send-receive sequences). Each IUCV interrupt (external interrupt X'4000') is described in a 40 byte interrupt buffer. QUERY-NEXT passes its relevant information to the user.

2. IUCV allows up to 65,535 paths for a communicator (MAXCONN in OPTION directory statement). However, CP supports only four connections (i.e. paths) when MAXCONN is not specified in the CP directory, while this package currently allows up to 2000 connections. There is also an IUCV directory statement to specify what kind of connections are accepted by CP and how many outstanding messages per path (MSGLIM parameter) are allowed. Virtual machines usually are not allowed to have connections to other virtual machines, and without specification of MSGLIM in the directory only ten messages per path may be outstanding. (The upper limit for this parameter is 255 for pre-ESA systems and 65,535 for ESA systems.) This package currently allows 10,000 as a maximum value. Thus, to make full use of the package, the CP directory must be changed. Discuss this topic with your system administrator.

3. Function names can be abbreviated to four characters in length, and up to eight characters of a name are tested. (For INIT and TERM the names INITIALIZE and TERMINATE are allowed.)

4. To make programming of RXIUCVFN easier, only the first letter of an input keyword is inspected. (IUCV('QUERY','V') and IUCV('QUERY','VIRTUAL') have the same effect as IUCV('QUERY','VERSION').)

5. The QUERY-NEXT deletes the external interrupt buffer when no action is needed. (Interrupt buffers of type 02, 03, 04 and 05 are always deleted, while for type 06 and 07 the buffer is only deleted when no reply is pending.) When an action is needed this action deletes the interrupt buffer. (A pending con-

nection type interrupt buffer is deleted when the corresponding path is accepted or severed. Other cases are treated in a similar fashion.)

6. When more interrupt buffers than specified in the INIT function are pending, the IUCV interrupts are temporarily disabled to avoid queue overflow. This event is reflected in the result of QUERY-STATUS. In this situation the pending interrupt buffer queue should be emptied.

7. With QUERY-NEXT the user is informed that a message needs a reply only when the whole message has been received. It was done this way primarily to distinguish between pending messages and replies not yet sent. If this solution is found unsatisfactory it may be changed.

8. When using the WAIT function on VM/SP the timer should be set to real (CP SET TIMER REAL) because otherwise the wait may never end.

9. The WAIT function has been generalized for version 1.02. This was primarily done because between an IUCV('QUERY','NEXT',,pathid) returning only the value 'pending' (no interrupt buffer pending on the specified path) and the actual IUCV('WAIT') another interrupt buffer may have arrived.

10. In some cases the message class (represented as integer value) is used for multiplexing. On the receiving side this information is available before an actual RECEIVE operation has been invoked. REXX is powerful enough to handle the message class value in any desired way.

11. The PRMDATA option in IUCV is only available in VM release 3.00 and later. Its use in earlier releases may cause strange and unrecoverable errors.

12. When initialized with the keyword 'CMS' or 'NATIVE', this package monopolizes the external new PSW. Thus, while IUCV is active (between INIT and TERM) no program should run using external interrupts. When a program is active using external interrupts other than '4000'x or '00xx'x before INIT is issued, there is no problem (clock comparator or VMCF programs). But they should not change the EXTNPSW between INIT and TERM. An IUCV('WAIT') also ends when other external interrupts occur and gives its code as a result.

13. When initialized with the 'APPLICATION' keyword, the external new PSW is not touched. In this case, the WAIT function only terminates on timer or IUCV interrupts. The WAIT function uses the WAITECB macro. (Make sure that the corresponding function is available on the VM system you use. It may not be provided on VM/SP release 4 and earlier.) Initializing REXXIUCV with the keyword 'APPLICATION' is the suggested method for application programs, if no monopolized vmid is needed and no interrupts other than timer and IUCV.

14. When initialized with the keyword 'CMS' or 'APPLICATION', RXIUCVFN supports waiting through the REXXWAIT package in addition to the function WAIT. The REXXWAIT program in the REXXWAIT package allows waiting for basic events such as console, message, mail and time events. In addition, other programs (such as REXXIUCV) can export their events.

15. Some parameters in the IUCV parameter list are not supported (FCNCD and MSGTAG are always set to hexadecimal zeros). The IUCV trace table handling is not used in this package.

16. The IPQUSCE option is not supported: A connection cannot be requested or accepted in quiesced mode.

17. Some IUCV functions (DESCRIBE, TESTCMPL, TESTMSG, SETMASK, SETCMASK) are not supported because their function is performed in a different way in this package. The IUCV QUERY is only partially used and supported in INIT.

18. The use of CMS IUCV is a local matter. It does not influence the communication partners with exception of the use of the user data field in CONNECT. E.g., if an application consists of a server and many clients, the server may or may not use CMS IUCV, and each client may or may not use it as well, independently of each other. For the use of CMS IUCV, see the sample CMS IUCV session below.

19. Connections to the CP service '*MSG' are of special interest. When using this package (see the REXX-EXEC 'QUSER' below), normal VM messages, warnings, and SMSG's can be received at the time the user wants to receive them. With the CP command 'SET' different objects (MSG, SMSG, WNG, IMSG, EMSG, CPCONIO and VMCONIO) can be directed to IUCV. It is easy to write an REXX-EXEC to stack the output of any CMS commands not touching the EXTNPSW, but there is the limit of 255 outstanding messages.

20. PVM (extended version 4.1001) supports internode IUCV communications. Such a connection is established when a connection to PVM has been requested with the actual target communicator specified in the 16 byte 'userdata' field of the IUCV parameter list (8 bytes nodeid and 8 bytes userid). PVM just stores and forwards corresponding IUCV connection requests, messages, replies and so on. There are some limitations: Internode connections cannot be quiesced and priority messages and replies are not allowed.

21. IUCV between different systems as supported by PVM (extended version 4.1001) and CMS IUCV make conflicting use of the userdata field in connection requests. RXIUCVFN version 1.09 supports both and therefore resolves this conflict. The vmid of the PVM virtual machine (usually 'PVM') can be specified in the field 'mvmid' of the INIT function to allow PVM IUCV and CMS IUCV together.

3.0 Functions to Activate or Deactivate IUCV

There are two functions to manipulate the IUCV environment, one to initialize or activate and one to terminate or deactivate IUCV, which must be initialized before paths and messages can be manipulated:

3.1 Function INIT

IUCV('INIT',numpath,[cmsiucv,[cmsname],[mvmid]],[numintb])

Result: numpath

The function returns the actual maximal number of paths allowed. This number may be smaller than the number requested in the input parameter, depending on the limit set in the CP directory.

In the code of this function an IUCV QUERY (to determine the actual MAXCONN parameter in the CP directory) and an IUCV DCLBFR (to specify the IUCV external interrupt buffer) are executed. Depending on whether native or CMS IUCV is initialized, DCLBFR is directly called or via HNDIUCV SET. Some buffers are allocated and an external interrupt handler may be installed by overwriting the EXTNPSW.

Note: QUERY-STATUS can be used to check whether an INIT has already been performed.

3.2 Function TERM

IUCV('TERM')

Result: pathlist

The function returns a list of pathids automatically severed by IUCV.

In the code of this function an IUCV RTRVBFR (to reset IUCV) is executed. The allocated storage is freed again and the initial EXTNPSW is properly reinstalled.

Note: If the RXIUCVFN package is unloaded before a TERM has been performed, an implicit TERM will be issued.

4.0 Functions to Manipulate IUCV Paths

There are five functions to manipulate IUCV paths. Paths can be connected, accepted, severed, quiesced and resumed.

4.1 Function CONNECT

IUCV('CONNECT',vmid,msglim,[prio],[userdata])

Result: pathid msglim

The function returns the pathid and the msglim value returned by IUCV.

In the code of this function an IUCV CONNECT (to request a path with the specified vmid) is executed. On the side of the vmid an IUCV external interrupt indicates that a connection is pending.

Note: All connections allow the IUCV 'PRMDATA' option by default.

4.2 Function ACCEPT

IUCV('ACCEPT',pathid,msglim,[prio],[userdata])

Result: pathid msglim

The function returns the pathid and the msglim value returned by IUCV.

In the code of this function an IUCV ACCEPT (to accept a requested path with the requesting vmid) is executed. On the side of the requesting vmid an IUCV external interrupt indicates that the connection is now complete.

Note: All connections allow the IUCV 'PRMDATA' option by default.

4.3 Function SEVER

IUCV('SEVER',pathid,[userdata])

Result:

The function returns no data.

In the code of this function an IUCV SEVER (to sever a requested or already established path) is executed. On the other side of the path an IUCV external interrupt indicates that the connection or connection request has been severed. (Pending messages on this path are purged by IUCV.)

Note: The 'IPALL' option is not supported so each path must be severed separately.

4.4 Function QUIESCE

IUCV('QUIESCE',pathid,[userdata])

Result:

The function returns no data.

In the code of this function an IUCV QUIESCE (to suspend incoming messages temporarily) is executed. On the other side of the path an IUCV external interrupt indicates that the connection has been quiesced.

Note: The 'IPALL' option is not supported so each path must be quiesced separately.

4.5 Function RESUME

IUCV('RESUME',pathid,[userdata])

Result:

The function returns no data.

In the code of this function an IUCV RESUME (to allow again messages on a quiesced path) is performed. On the other side of the path an IUCV external interrupt indicates that the connection has been resumed.

Note: The 'IPALL' option is not supported so each path must be resumed separately.

5.0 Functions to Manipulate IUCV Messages

There are five functions to manipulate IUCV messages. On the source side, messages can be either sent or purged. On the target side, messages can be either received or rejected. When the source side has indicated that a reply is wanted, the target side can send a reply.

5.1 Function SEND

IUCV('SEND',pathid,data,srccls,trgcls,[prio],[prm],[rpl,[rpllength]])

Result: msgid

The function returns the msgid (which is needed to purge a message or to map messages, message complete interrupts, and replies).

In the code of this function an IUCV SEND (to send the message over a specified path) is executed. The target vmid gets an external interrupt with a type 08 (pending priority message) or a type 09 (pending non-priority message).

Note: The length of a message or a reply is limited by RXIUCVFN to 100,000 bytes.

5.2 Function PURGE

IUCV('PURGE',pathid,srccls,msgid)

Result:

The function returns no data.

In the code of this function an IUCV PURGE (to purge the message which has been sent) is executed. If the message has not yet been received on the target side, a later RECEIVE will give a return code 9.

Note: A message not yet received uses storage on the senders side. To free (DMSFRET) this space in case the target side is not willing to receive or reject the message, the clean way is to purge the message.

5.3 Function RECEIVE

IUCV('RECEIVE',pathid,trgcls,[msglength])

Result: data

The function returns the message (or part of it if 'msglength' is too short).

In the code of this function an IUCV RECEIVE (to receive the message) is executed unless the message is contained in the parameter list ('PRMDATA' option). Receiving a message in parts produces a return code 5 until the whole message has been received. If the message is fully received (in one or more receive steps)

and if no reply is needed, the sender will be informed with a pending (non-)priority message completion interrupt.

Note: This package does not allow the reception of messages with the same target class on the same path out of order, but priority messages come ahead of all nonpriority messages.

5.4 Function REJECT

IUCV('REJECT',pathid,trgcls)

Result:

The function returns no data.

In the code of this function an IUCV REJECT (to reject the message) is executed unless the message is contained in the parameter list ('PRMDATA' option). When needed, an empty reply is generated by IUCV.

Note: The sender of a message cannot distinguish whether the message was received or rejected.

5.5 Function REPLY

IUCV('REPLY',pathid,data,trgcls,[prio],[prm])

Result: msgid

The function returns the msgid.

In the code of this function an IUCV REPLY (to reply to a message) is executed. The sender gets an external interrupt indicating message complete.

Note: The length of the reply may not exceed the length specified by the sender except when the reply is sent with PRMDATA option. In this case, the length of the reply may not exceed 8.

6.0 Auxiliary Functions

There are two other functions needed to use this package. A query function gives the user some information (e.g. describes the relevant content of the pending interrupt buffers) and a wait function allows the user to enter a wait state.

6.1 Function QUERY

IUCV('QUERY',qkeyword,[others])

Result: others

The parameter qkeyword is 'STATUS', 'CMSNAME', 'MVMID', 'VERSION', 'PATH', 'NEXT' or 'REPLY'. The further input parameters "others" depend on the qkeyword, as shown below. The function returns data dependent on the qkeyword.

Note: The query function is allowed before INIT.

IUCV('QUERY','STATUS')

Result: init error intstatus

where 'STATUS' is a keyword. The function returns the three items 'init', 'error' and 'intstatus' in the return string.

IUCV('QUERY','CMSNAME')

Result: cmsname

where 'CMSNAME' is a keyword. The function returns the current CMSIUCV name.

IUCV('QUERY','MVMID')

Result: mvmid

where 'MVMID' is a keyword. The function returns the current monopolized vmid.

IUCV('QUERY','VERSION')

Result: name version date

where 'VERSION' is a keyword. The function returns name ('RXIUCVFN'), version (currently 1.36) and date of the package.

IUCV('QUERY','PATH',[pathid])

Result: pathlist | [pathid vmid]

where 'PATH' is a keyword. When no pathid is specified, the function returns the list of all current pathids. Otherwise it returns the pathid and the corresponding vmid, when available.

IUCV('QUERY','NEXT',[type],[pathid],[msgcls])

Result: pending [type pathid others]

where 'NEXT' is a keyword. The function always returns the number of pending interrupt buffers and, when available, type, pathid and other information depending on the type:

01 Pending connection:
msglim vmid prio userdata
02 Connection complete:
msglim userdata
03 Path has been severed:
userdata
04 Path has been quiesced:
userdata
05 Path has been resumed:
userdata
06 Pending priority message completion:
msgid srccls [rpl rpllength]
07 Pending nonpriority message completion:
msgid srccls [rpl rpllength]
08 Pending priority message:
msgid trgcls msglength [rpl rpllength]
09 Pending nonpriority message:
msgid trgcls msglength [rpl rpllength]

Note: As an output from QUERY-NEXT, 'userdata' is always contained in the last 18 bytes of the result string. It consists of 16 bytes actual data enclosed by '<' and '>' to allow easier parsing in REXX. The output 'prio' for a pending connection is either 'Pr' or 'Nonpr'. The result item 'rpl' is 'Reply'.

IUCV('QUERY','REPLY',pathid,[srccls])

Result: data

where 'REPLY' is a keyword. The function returns the reply data.

6.2 Function WAIT

IUCV('WAIT',seconds,[how])

Result: extint

The parameter 'how' indicates whether a wait state should be entered even when interrupt buffers are already pending. Its value can either be a keyword or a number. As a number it indicates how many interrupt buffers (at least) have to be pending to enter no wait state. The function returns a hexadecimal string four characters in length describing the interrupt causing the end of the wait:

0000 Interrupt from the console (or other I/O interrupts)
0080 The timer expired
4000 IUCV interrupt buffer pending
00xx Other external interrupt type (CP command EXTERNAL xx)

For all other external interrupts the new external PSW is loaded, which was in EXTNPSW before it was overwritten by the INIT function.

VM/SP uses the interval timer in the STIMER and TTIMER macros which results in a '0080' external interrupt. RXIUCVFN version 1.11 also supports VM/XA where the clock comparator is used instead of the interval timer resulting in '1004' external interrupts. However, an external interrupt of type '0080' is reported also on VM/XA systems when the timer expires.

The parameter 'how' is used to control waiting in presence of pending IUCV interrupt buffers. A positive number allows preventing already pending IUCV interrupt buffers from causing the WAIT function to terminate. In other words, the REXX program can wait for the next IUCV event in spite of a number of pending IUCV interrupt buffers, which it decided temporarily not to handle. This feature avoids a nonproductive busy loop around the WAIT function. The following table shows whether the WAIT function immediately terminates depending on 'pending' (as reported by the QUERY-NEXT function) and 'how':

how → / pending ↓	NOWAIT	WAIT	n	n+1
0	No	No	No	No
n	Yes	No	Yes	No

Note: The WAIT function terminates whenever an interrupt occurs in the virtual machine. Especially, if a REXX-EXEC using RXIUCVFN with CMS IUCV runs together with another IUCV program using CMS IUCV, WAIT also reports IUCV interrupts for the other program. In other word, WAIT may terminate with the result '4000' while QUERY-NEXT does not show pending interrupts.

7.0 Support for REXXWAIT

REXXWAIT (also called REXX/WAIT) provides access to a central wait function. (See REXXWAIT SCRIPT for more information.) When the REXXWAIT MODULE is available, RXIUCVFN supports its functions. Especially, combinations of the WAIT and QUERY-NEXT functions of RXIUCVFN can be replaced by calls of the WAIT function of REXXWAIT.

7.1 The REXX Function WAIT

WAIT(...,'IUCV' ['TYPE' type] ['PATH' pathid] ['MSGCLS' msgcls],...)

Result: rc 'IUCV' type pathid others

The function call waits for a pending interrupt buffer matching the type, pathid and message class information specified (if no one is already pending) and returns a string similar to the result of the QUERY-NEXT function, but with the keyword 'IUCV' as the first token and without the number of pending interrupt buffers. Waiting for IUCV events can be restricted to a specific type, path and/or message class. The keyword 'IUCV' comes from the way REXXWAIT reports events. More than one event for RXIUCVFN can be entered in one call of the WAIT function:

```
res = WAIT('IUCV PATH 7','CONS','IUCV TYPE 3 PATH 5','IUCV','TIME 10S')
```

is legal. Since REXXWAIT processes the events in the sequence entered, any IUCV event for pathid 7 is reported to the REXX program before a console, a path 5 severed, or another IUCV event (assuming no explicit defaults have been set using the SETVALUE function) when some of these events are already pending. The timer-expired event comes last and is only reported when no other event is pending or has happened before the timer has expired.

Note: Waiting for 'IUCV' events through REXXWAIT is only enabled when RXIUCVFN is initialized for CMS IUCV using the method 'Cms' or 'Application' in the INIT function. However, 'Application' cannot be used together with a monopolized vmid due to the way RXIUCVFN resolves the incompatibility between CMS IUCV and PVM IUCV.

In addition to the values defined by REXXWAIT, the function returns the following return codes:

10 No pending interrupt buffer with buffer pool overflow

When waiting is restricted to specific type, pathid or message class, no corresponding interrupt buffer is pending, but IUCV interrupts are disabled because of buffer pool overflow, the function terminates with a return code 10 to make the program aware of this fact. (This may not always be what a REXX programmer expects. A call of the form `WAIT('IUCV PATH 7','IUCV')` may terminate with a return code 10 because of the first argument and despite the second argument.)

7.2 The REXX Function SETVALUE

SETVALUE('IUCV' ['TYPE' type] ['PATH' pathid] ['MSGCLS' msgcls])

Result: rc olddefaults

This function call sets the default type, pathid and/or message class for waiting on IUCV events. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the defaults are 'TYPE * PATH * MSGCLS *'.

7.3 The REXX Function QUERYVALUE

QUERYVALUE('IUCV VERSION')

Result: rc name version date

The function call returns the same result as the QUERY-VERSION function and can be used instead of it.

QUERYVALUE('IUCV DEFAULTS')

Result: rc defaults

This function call returns the current default settings for type, pathid and message class.

7.4 The REXX Function RESETVALUE

RESETVALUE('IUCV')

Result: rc

This function call resets the defaults for type, pathid and message class, and returns no data.

8.0 Examples

To give potential users an idea of how to use the functions of this package, a sample of an IUCV communications pattern and two examples are attached. The first example demonstrates how to make use of IUCV in a terminal session, while the second example shows a REXX-EXEC making use of IUCV to receive VM messages from RSCS through a path with the CP service '*MSG'.

8.1 IUCV Communications Pattern

Figure 3 shows a possible communication trace in visual form involving a server and two clients. The two clients connect to the server and send data.

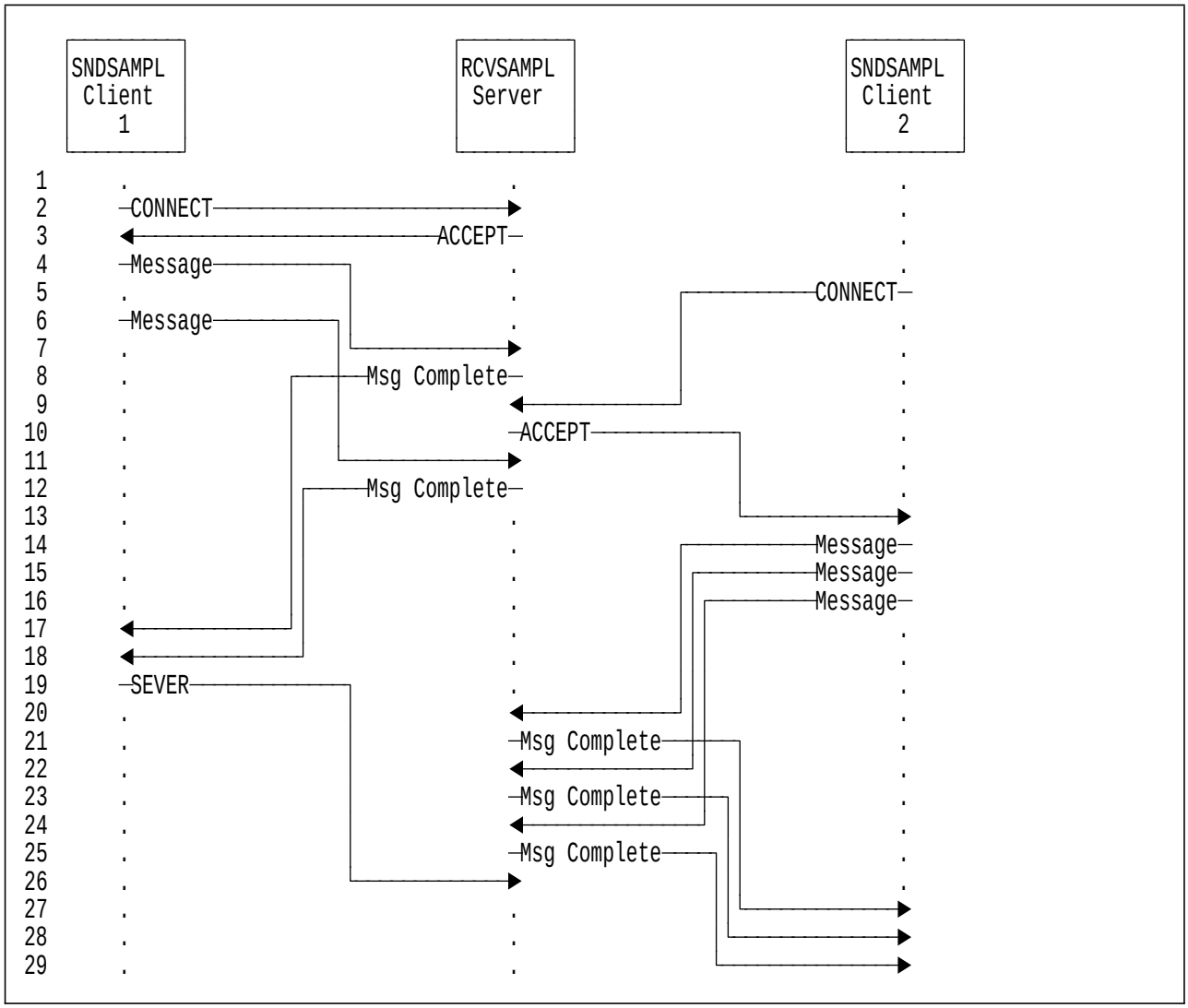

Figure 3. Sample IUCV communications pattern

In line 2 and 3, the path between client 1 and the server gets established with the two functions CONNECT and ACCEPT. The arbitrary 'Message' (e.g. in line 4) gets sent through the SEND function, and the 'Msg Complete' (e.g. in line 8) gets sent when the server issues the RECEIVE function (for one-way communications) or the REPLY function (for two-way communications). When client 1 is done, it issues the SEVER function. Also the server will issue the SEVER function to completely release the path, but this is not shown because it does not get forwarded to the client. (At the end of the trace, the path between client 2 and the server is still established.)

8.2 Sample of an Interactive Session

Using the following small REXX-EXEC 'IUCV'

```
/* IUCV ------------------------------------------------------------*/
trace o
parse arg argstring
interpret 'response =' argstring
say response
exit rc
```

the functions of the package can be used to work interactively with IUCV. In the following session both sides of the IUCV connection were played on the same userid. To distinguish these two sides the linear session is structured with horizontal offsets (INIT and TERM are somewhere in between).

```
               iucv iucv(init,2)
               2
               R;
iucv iucv(connect,'RFH',255,'PRIO')
0 255
R;
                               iucv iucv(query,next)
                               1 1 1 255 RFH Pr <               >
                               R;
                               iucv iucv(accept,1,255,'PRIO')
                               1 255
                               R;
iucv iucv(query,next)
1 2 0 255 <               >
R;
iucv iucv(send,0,'Hello',2,3)
2304
R;
                               iucv iucv(query,next)
                               1 9 1 2304 3 5
                               R;
                               iucv iucv(receive,1,3)
                               Hello
                               R;
iucv iucv(query,next)
1 7 0 2304 2
R;
iucv iucv(query,next)
0
R;
iucv iucv(sever,0)

R;
                               iucv iucv(query,next)
                               1 3 1 <               >
                               R;
                               iucv iucv(query,next)
                               0
                               R;
                               iucv iucv(sever,1)

                               R;
               iucv iucv(term)

               R;
```

This sample session may give potential users the idea that IUCV can be used very easily with that package. But one should not forget that IUCV connections produce asynchronous events and we poor mortals are trained to think sequentially and not in parallel.

8.3 Sample CMS IUCV Session

The connection setup for an interactive IUCV session using CMS IUCV is shown. The REXX-EXEC 'IUCV' has been modified to redisplay the argument string, and argument string and result string are both displayed with a time stamp in front.

```
iucv iucv('INIT',10,'CMS','SERVER  ')
09:27:04 iucv('INIT',10,'CMS','SERVER  ')
09:27:04 10
R;
                    iucv iucv('INIT',1,'CMS','CLIENT  ')
                    09:27:22 iucv('INIT',1,'CMS','CLIENT  ')
                    09:27:22 1
                    R;
iucv iucv('WAIT',3600)
09:27:34 iucv('WAIT',3600)
                    iucv iucv('CONNECT','RFH',255,'N','SERVER  ')
                    09:27:53 iucv('CONNECT','RFH',255,'N','SERVER  ')
                    09:27:53 0 255
                    R;
09:27:53 4000
R;
iucv iucv('QUERY','NEXT')
09:28:09 iucv('QUERY','NEXT')
09:28:09 1 1 0 255 RFH1 Nonpr <SERVER  ........>
R;
                    iucv iucv('WAIT',3600)
                    09:28:20 iucv('WAIT',3600)
iucv iucv('ACCEPT',0,255,'N','Anything')
09:28:57 iucv('ACCEPT',0,255,'N','Anything')
09:28:57 0 255
R;
                    09:28:57 4000
                    R;
                    iucv iucv('QUERY','NEXT')
                    09:29:08 iucv('QUERY','NEXT')
                    09:29:08 1 2 0 255 <Anything........>
                    R;
```

The userid 'RFH' initializes CMS IUCV under the name 'SERVER'. The client userid 'RFH1' also uses CMS IUCV and initializes under the name 'CLIENT'. To start a connection with the server, the client calls CONNECT with the servers CMSNAME (i.e. 'SERVER') in the first eight bytes of the user data field.

Note: RXIUCVFN pads the user data field with X'00'. Therefore, the CMSNAME of the server must be padded with blanks in CONNECT. Otherwise, CMS IUCV severs the connection request because 'SERVER' padded with blanks and 'SERVER' padded with X'00' is not the same.

8.4 The REXX-EXEC QUSER

The well-known CMS EXEC 'NQUERY' can be used to query the status of one or more computer users on the same computer or on other computers (VM installations only) connected via the RSCS network. The answer is displayed on the invoker's terminal as a VM message. The following REXX-EXEC receives these messages via an IUCV connection with the CP service '*MSG' and allows the answer to be stacked for use in other EXEC's. (Only the query of the status of one computer user is supported and no use of the nickname facility is made.)

```
/* QUSER -------------------------------------------------------*/
trace o

parse arg argstring
argstring = strip(argstring)
if substr(argstring,1,1) = '?' then do
   say 'Use the "QUSER" command to query the status of a computer user'
   say 'on your computer  or on other computers  that are connected to'
   say 'your computer via the RSCS network.  The format of the command'
   say 'is:                                                          '
   say '    QUSER userid <AT <nodeid>> <(<Timeout timeout> <STack><)>>'
   say ' Default:            yournode            10 (seconds)        '
   exit 100
end

/* Who am I, anyway?                                                 */
address command 'IDENTIFY ( LIFO'
parse upper pull userid . locnode . rscsid .

/* Split arguments into parameters and options                       */
parse upper var argstring parameters '(' options ')' rest

/* Parse the parameters                                              */
parse var parameters quser at qnode rest
if quser='' then call error 24 'No names specified'
if at¬='AT' & at¬='' then call error 24 'Invalid parameters specified'
if rest¬='' then call error 24 'Invalid parameters specified'
if qnode='' then qnode = locnode
if length(quser)>8 then call error 24 'Invalid user' quser
if length(qnode)>8 then call error 24 'Invalid node' qnode

/* Parse the options                                                 */
timeout = 10
stack = 0
do forever
   parse var options token options
   select
      when token='' then leave
      when abbrev('STACK',token,2)=1 then stack = 1
      when abbrev('TIMEOUT',token,1)=1 then do
         timeout_error = '20 Invalid timeout'
         parse var options timeout options
         if datatype(timeout)¬='NUM' then call error timeout_error
         timeout = format(timeout,,0)
         if timeout<1 | timeout > 3600 then call error timeout_error
      end
      otherwise call error 20 'Invalid option'
   end
end

/* Initialize the response line etc.                                 */
line = ''
others = 0

/* Load RXIUCVFN and initialize IUCV if needed                       */
if qnode¬=locnode then do
   address command 'RXIUCVFN TEST'
   notloaded=rc
   select
      when notloaded=0 then nop
      when notloaded=1 then address command 'RXIUCVFN LOAD'
      otherwise call error 45 'Problems with loading RXIUCVFN'
   end
```

```
   iucverror = 0
   iucvresponse = IUCV('QUERY','STATUS')
   if rc¬=0 then iucverror = 1
   if ¬iucverror then do
      parse upper var iucvresponse init .
      select
         when (init='YES' | init='CMS') then iucvinit = 1
         when init='NO' then do
            iucvinit = 0
            iucvresponse = IUCV('INITIALIZE',1)
         end
         otherwise iucverror = 1
      end
   end
   if ¬iucverror then do
      iucvresponse = IUCV('CONNECT','*MSG',255)
      if rc=0 then parse upper var iucvresponse pathid .
      else iucverror = 1
   end
end

/* Make the appropriate query for a local request                    */
if qnode=locnode then do
   address command 'MAKEBUF'
   address command 'EXECIO 1 CP (LIFO STRING QUERY USER' quser
   if queued()>0 then parse pull line
   address command 'DROPBUF'
end

/* Make the appropriate query for a remote request                   */
if qnode¬=locnode then do
   if ¬iucverror then do
      address command 'MAKEBUF'
      address command 'EXECIO 1 CP (LIFO STRING QUERY SET'
      if queued()>0 then parse pull msg savemsg .
      if msg¬='MSG' then savemsg = 'ON'
      savemsg = left(savemsg,4)
      address command 'DROPBUF'
      address command 'CP SET MSG IUCV'
      address command 'CP SMSG' rscsid 'CMD' qnode 'CPQ USER' quser
      found = 0
      wait = 1
      do while(¬found & ¬iucverror)
         extint = IUCV('WAIT',timeout,wait)
         if rc¬=0 then iucverror = 1
         if extint¬='4000' then leave
         iucvresponse = IUCV('QUERY','NEXT',,pathid,1)
         parse upper var iucvresponse cpending ctype .
         if cpending>0 then do
            select
               when ctype=2 then nop
               when ctype=3 then iucverror = 1
               when ctype=9 then do
                  fromuser = IUCV('RECEIVE',pathid,1,8)
                  if rc¬=5 then iucverror = 1
                  fromuser = strip(fromuser)
                  message = IUCV('RECEIVE',pathid,1)
                  if rc¬=0 then iucverror = 1
                  message = strip(message)
                  if fromuser=rscsid then do
                     parse var message text rest
                     if left(text,6)='DMTRGX' then message = rest
                     Parse var message next rest
```

```
                  if next='FROM' then do
                     parse var rest fromid rest
                     parse var fromid msgnode '(' msguser '):' etc
                     if msguser¬='' then do
                        fromuser = msguser 'at' msgnode
                        message = rest
                     end
                  end
               end
               else fromuser = fromuser 'at' locnode
               if fromuser=rscsid then found = 1
               parse var message . ' CPQ: ' line
               if fromuser¬=rscsid | line='' then do
                  say 'Message from' fromuser':'
                  say message
                  others = 1
               end
            end
            otherwise nop
         end
      end
      wait = IUCV('QUERY','NEXT',0) + 1 /* no valid type  */
   end
   address command 'CP SET MSG' savemsg
   do forever /* because we do not want to loose anything */
      iucvresponse = IUCV('QUERY','NEXT',9,pathid,1)
      parse upper var iucvresponse cpending ctype .
      if ctype='' then leave
      fromuser = IUCV('RECEIVE',pathid,1,8)
      fromuser = strip(fromuser)
      message = IUCV('RECEIVE',pathid,1)
      message = strip(message)
      say 'Message from' fromuser':'
      say message
      others = 1
   end
end
end

/* Terminate IUCV and unload RXIUCVFN if needed                      */
if qnode¬=locnode then do
   if ¬iucverror then iucvresponse = IUCV('SEVER',pathid)
   if ¬iucverror & ¬iucvinit then iucvresponse = IUCV('TERM')
   if notloaded then address command 'NUCXDROP RXIUCVFN'
   if iucverror then call error 50 'IUCV problems'
end

/* Give the user an answer                                           */
parse var line text rest
if left(text,6)='DMKCQY' then line = rest
if others then say ' '
if qnode¬=locnode & line='' then do
   say 'The appropriate answer may arrive later as a message!'
   select
      when extint='4000' then call error 60 'No correct message found'
      when extint='0080' then call error 60 'Timeout occurred'
      otherwise call error 60 'Other external interrupt stopped query'
   end
end
if stack then queue '*' line
else say line
exit 0
```

```
/* Error message and exit routine                                  */
error: parse arg return_code error_message
   say '***  Error  ***' error_message
   exit return_code
```

Many local changes in CP and RSCS programs make the life of such EXEC's very difficult. RSCS messages describing an event (such as 'node xyz not connected') can vary from node to node. Another problem of the above EXEC is the fact that other VM messages can arrive while the EXEC expects one specific VM message from RSCS so that a 'sophisticated' analysis is needed.

8.5 Further Examples

The two REXX programs RISERVER EXEC and RICLIENT EXEC provided with the REXXIUCV package show a simple client-server application. The server code (RISERVER) sends information to as many clients (RICLIENT) as connect. The server accepts connection requests as long as there is a free path to use. The clients connected at the same time are serviced in parallel.

The REXX Handbook by Gabriel Goldberg and Philip H. Smith III (McGraw-Hill, and IBM form number SB20-0020) contains a chapter about REXXIUCV in which another simple client-server application is presented. Especially, robustness and economical resource use are discussed.

Index