# RXPIPE for z/OS

A free alternative to IBM's PIPE command

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>General information</td>
<td>3</td>
</tr>
<tr>
<td>Starting RXPIPE</td>
<td>4</td>
</tr>
<tr>
<td>Simple program – format a TSO LISTCAT</td>
<td>5</td>
</tr>
<tr>
<td>Read from a dataset</td>
<td>6</td>
</tr>
<tr>
<td>Write to a dataset</td>
<td>7</td>
</tr>
<tr>
<td>Using the user's stack</td>
<td>8</td>
</tr>
<tr>
<td>The FILTER and BUILD commands</td>
<td>9</td>
</tr>
<tr>
<td>Format output – list</td>
<td>10</td>
</tr>
<tr>
<td>Format output – tables</td>
<td>11</td>
</tr>
<tr>
<td>Beyond the stack – save and load</td>
<td>12</td>
</tr>
<tr>
<td>Beyond the stack – fanout and fanin</td>
<td>13</td>
</tr>
<tr>
<td>Looping over the stack – FOREACH</td>
<td>14</td>
</tr>
<tr>
<td>Using an external function</td>
<td>15</td>
</tr>
<tr>
<td>Some internals</td>
<td>16</td>
</tr>
<tr>
<td>Other bits and pieces</td>
<td>17</td>
</tr>
</tbody>
</table>
Introduction

Who am I

My name is Willy Jensen, I have been working with large IBM systems since March 1973, in various capacities, starting as a humble systems operator and ending in December 2015 as system engineer at the Swiss bank UBS (at the time the 7th largest bank worldwide). All that time I worked with the base operating system and utilities, starting with VS1 and all the way up to z/OS. After my retirement I have been lucky to have access to a z/OS, so I continue working on my tools, and get a peek at the system internals.

I can be reached through my email willy@harders-jensen.com, my public domain programs are at https://harders-jensen.com

RXPIPE background

Ever since the first time I encountered the pipe feature in Ms DOS, and later in Linux, I have been envious that we did not have something similar in MVS, as it was then. The simplicity of a number of small programs strung together to form a whole, rather than a big program trying to anticipate all a user’s needs. Then a couple of years ago I was part of a VM-to-z/OS conversion, which involved a lot of converting CMS REXX programs with heavy use of the CMS PIPE command. While not as simple as the Linux piping, the major difference, I felt, was that you had to add the word ‘PIPE’ in front, which I can live with. The PIPE command is included in CMS, you can get the PIPE command for z/OS, but it is part of the Batch Pipes product and is thus costly, and not something you can rely on being available anywhere. I thought about this for quite some time, a couple of years really, till I decided to give it a go. And eventually I wrote RXPIPE.
General information

RXPIPE is not a replacement for the IBM PIPE command, rather it is an attempt to implement what I find the most useful elements of the PIPE command, in a small and free TSO/REXX based package.

The base product uses only standard TSO and ISPF features, though some add-ons are recommended.

RXPIPE executes a series of commands, called stages, passing the output of each stage as input to the next, via the TSO stack. In this it works similar to a MSDOS or Linux pipe. RXPIPE has a number of built-in transformers to manipulate the contents of the stack but will also accept external programs as stages. The stages are separated by the vertical bar (|). This might a problem in some cases, i.e. when using a REXX expression with boolean OR, but in such cases, you can put a backslash (\) character before the vertical bar. The backslash is dropped, and the vertical bar is passed on as part of the command.

Any command and program can be part of a stage, as input and output can be the stack (preferred) or a dataset, which could have been generated by a previous stage. Parameters for a program can be generated by a previous stage. Some stuff, like WTOs and TPUTS cannot be trapped, but that is a TSO restriction and not something that RXPIPE can do anything about.

The callers stack may be used as input and the final stack may be written back to the callers’ stack. The internal stack is listed and dropped by default at the end of RXPIPE processing.

RXPIPE has currently 70+ built-in commands, of which 8 are separate members, mostly due to their size.

Some of the command names can be abbreviated, sub-commands typically cannot. Command names are not case sensitive.

Command operands and expressions are case sensitive unless otherwise stated for the command. Some of the commands are similar to the IBM PIPE commands, though may have a different name. The syntax is simple, with no parentheses or equal signs. Parameters must be quoted if they would otherwise cause ambiguity (i.e. WHERE is a parameter for some commands).

RXPIPE executes with TSO PROFILE NOPREFIX, so datasets must be fully qualified - quotes are optional.

RXPIPE in general do not care about return codes, only with failed commands. You will have to use the IF or SET commands to deal with exceptional return codes.

Variables are accepted in commands. A variable is a name starting with an ampersand (&) and terminating in something which is not a character or a number. Lone ampersands are ignored as they could be the AND operand in an expression. You can use variables defined by the the SET command or the built-in variables.

RXPIPE is built with focus on functionality, not so much on performance. It is expected, as time goes by, that commands will be reviewed and optimized.
Starting RXPIPE

RXPIPE can be started in a number of ways, depending on whether is is started from a REXX pgm, or from
the TSO command line.

If started from a REXX you can do
   cc=RXPIPE(command)
   Call RXPIPE 'command'
   Address TSO "RXPIPEcommand"

If started from a CLIST you can only do  RXPIPE command

From from the TSO command line: RXPIPE command

From from the ISPF TSO command line: RXPIPE command

From from the ISPF command line: TSO RXPIPE command

RXPIPE is a REXX program, which means that you can start is as any other REXX pgm. The recommendation
is that it is installed as part of the start-up concatenation, but there is a script included in the package to do
TSO ALTLIB and ISPF LIBDEFs for the RXPIPE pgmlib. Due to the RXPIPE naming standard there shouldn’t be
any naming conflicts with other programs or applications.

General syntax

The general syntax is

   RXPIPE | command1 | ... | commandn

Each command consists of the command verb and optional parameters. The syntax is relaxed, there are no
parentheses nor equal signs, verbs and parameters are mostly case ignorant and text must only be In
quotes if there otherwise would be a clash between the text and parameters.

The downside of this is that the syntax checking might not catch bad parameters.
Simple program – format a TSO LISTCAT

This simple demo shows how you can run a TSO command and format the output from that.

The LISTCAT command is the standard TSO command. The EXClude command removes all lines containing the text ‘ACCOUNT-INFO’. The SPLIT command splits the lines at 2 blanks. The STRIP B command strips leading and trailing blanks (B=Both). The remaining lines are automatically listed when RXPIPE ends.

Format listcat – simple demo

program

RxPipe listc ent('DVL.PP.RXPIPE.V007.LIB') vol | excl ACCOUNT-INFO | split ' ' | strip b

Notes
• LISTC is an external command.
• Text for the EXClude command must match data case
• The stack is listed automatically at end

by Harders-Jensen IT
Reading is made simple, no need to allocate the dataset beforehand, or free it afterwards. You can also read a previously allocated dataset by DDname and a member from a partitioned dataset previously allocated by its DDname plus membername. And unix pathes too.

Essentially everything that BPXWDYN can allocate and EXECIO can read is supported.

The parameters are those that you would use for the BPXWDYN command, as that is what is used under the covers.

The first INCLUDE select all records from the APF section, the second INCLUDE selects specific volser. Note the use of the ‘caps’ operand.

Finally, all occurrences of ‘APF ADD’ are replaced by ‘APF’ to produce a nicer report.

You can read from a VSAM KDS or ESDS using the VSAM command. Only standard features (those allowed by the TSO REPRO command) are used, so it is not particular efficient, but it works. For more advanced VSAM access you should look at my RXVSAMBA program in CBT file 668, or at my website.
Write to a dataset

Program 1 – specific dataset options

```
RxPipe
| tsons del &$user..users.list
| lu *
| inc USER=
| write &$user..users.list new
    recfm(v,b) lrecl(255) blksiz(0)
    space(1,1) tracks
```

Notes
- TSONS executes the command, but does stack the response.
- The write command uses the &$USER variable for userid.
- UNIT(SYSDA) is defaulted.

Program 2 – automatic dataset options

```
RxPipe
| lu *
| inc USER=
| write &$user..users.list cond
```

Notes
- The COND operand means allocate with disp=SHR for an existing datasets, otherwise create a new dataset.
- The stack data is analyzed to get the record length and space requirements.

Writing is made simple, you don’t necessarily need to allocate or create the dataset beforehand, or free it afterwards.

Everything that BPXWDYN can allocate and EXECIO can write is supported.

The parameters are those that you would use for the BPXWDYN command, as that is what is used under the covers.

You can append to a dataset using the WRITE .. APPEND command (>>, this also works for pds members.
The DELETE command could have been used directly, but that would have written the response to the stack, which does not matter here because of the following INCLUDE command, just something to keep in mind.

You can specify the allocation options for a new dataset, or let RXPIPE determine them for you. The default recfm and lrecl for an automatic dataset is VB 27994, the space will be determined by the stack contents. The COND parameter tells RXPIPE to allocate the dataset if it does not exist, or use the existing one.

Member statistics are added if ISPF is active.

You can write to a VSAM KSDS or ESDS using the VSAM command instead of the WRITE command. Only standard features are used, so it is not particular efficient, but it works. For more advanced VSAM access you should look at my RXVSAMBA program in CBT file 668, or at my website.
Using the user’s stack

The stack as input and output

Program
say '->Stack as input '               ...            
Howdy folks
->Stack as input and output
Stack#: 2 ...              
Kilroy                     
was here

You can feed your external stack to RXPIPE and retrieve the RXPIPE internal stack back to the calling program. If the RXPIPE command starts with the vertical bar then your stack is read. Likewise if the RXPIPE command ends with the vertical bar then the RXPIPE stack is written. This demo shows you how.

The ‘text’ command puts a line on the stack.

The sub-function ‘ListStack’ shows the contents of the users’ stack.

The code in last section reads the user’s stack, does some manipulation (split the record), and writes it back.
The FILTER and BUILD commands

When a simple INCLUDE is not enough....

Program to show NONVSAM datasetname and volser

RxPipe
| listc lvl(sysa) nonvsam vol
| filter pos('NONVSAM -',r)>0 \ pos('VOLSER -',r)>0
| split 'DEVTYPE -'
| exc '-----X'
| combine 'VOLSER -'
| replace ' - ' '
| build left(w2,44) w4

Notes
• The BUILD command is used with the internal parsed variables ‘w2’ and ‘w4’ to finally format the output.

We looked briefly at the INCLUDE command. It passes the stack record if the text in the command matches, aka the POS function, though it is slightly more capable.

The FILTER command, however, is transformed into a REXX IF statement, so can be as simple or as complex as it needs to be. It also means that the syntax must adhere to REXX rules and restrictions. As previously mentioned, the vertical bar (Boolean OR) must be preceded with a backslash. When the filter expression is true then the stack record is kept, otherwise it is dropped.

The BUILD command is transformed into a REXX assignment statement, so have the same caveats as the FILTER command.

The demo shows how some of the internal variables can be used. When a record is pulled from the stack, it is parsed to individual words assigned to variables ‘w1’ to ‘w20’, plus ‘r’ containing the entire record.

Note the EXClude command, it works opposite to the INCLUDE command.

The demo introduces the COMBINE and REPLACE commands.
Listcat is rather complicated to format nicely, but it can be done. The challenge is how dashes ‘-‘ are used.

The demo shows the normal output from a LISTCAT, the intermediate listing and the final tabular list. The intermediate listing shows how the data must be formatted as discrete words, with the number matching the ‘cols’ statement of the TABLE command. The BUILD command is used to ensure that there are 2 words in each record, also in header records. The special character ‘#’ could be removed later by a final REPLACE.

Note that the dataset is unquoted and fully qualified.

The final ‘list’ command is not really necessary.
Format output – tables

Program 1 – horizontal by words

RxPipe
text Alpha Bravo Charlie Delta Echo Foxtrot Golf Hotel
| table col=3

Program 2 – vertical by words

RxPipe
text Alpha Bravo Charlie Delta Echo Foxtrot Golf Hotel
| table v col=def 10 c10 r8 bord all

Program 3 – by record

RxPipe
text Donald Duck, 14 A Street, Ducktown, 123-456-789
| text Bat Man, 34 Mansion Ave, Gotham City
| text Me And You, 33 Lovers Lane, Heartland, 798-456-123
| table r col, bord all

Notes
• Format automatically or fixed, with or without borders.

The TABLE command formats the columns according to the words or records that will be used to fill it, though you can specify column or page width yourself, as well as border options.

You can control the characters that are used to create the borders, both inner and outer, as well as prevent top- and bottom borders. The latter is useful if you want to concatenate tables.

The ‘text’ command adds the test data to the stack.
Beyond the stack – save and load

SAVE and LOAD

SAVE and LOAD is useful i.e if you wish to avoid multiple READS

Program which does 2 reports after just one read

RxPipe
< SYS1.PARMLIB(PROG00)
| save
| list 'APF where vol is DATA02'
| load clear
| list 'LNK where dsn is ADCD*'

Notes
• You must quote the text in the LIST command if part of the text contains the keyword 'where'.
• The plus sign (+) in the TEXT command creates a blank line.
• You can use a numeric id for the SAVE and LOAD commands, so you can have multiple temp stores. Default is 1.
• LOAD will by default add to the stack, hence the CLEAR option.

Just one stack may not always be sufficient. While you can have only one stack active, and cannot switch between stacks, the stack can be saved and loaded.

SAVE
• Stores a copy of the stack in a stem. You can provide an id if you want to save multiple times. The default id is ‘1’.
• The stack is retained unless you use the CLEAR keyword.
• You can add to a saved stream by the APPEND keyword.
• You can filter what is being saved by using the WHERE clause.

LOAD
• You can specify which stream id to load.
• You can load multiple streams either sequentially or rotated.
• You can clear the stack before load by using the CLEAR keyword.
• You can filter what is being loaded by using the WHERE clause.
Beyond the stack – fanout and fanin.

<table>
<thead>
<tr>
<th><strong>FANOUT</strong> + LOAD + implicit LIST</th>
<th><strong>FANOUT</strong> + FANIN + implicit LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxPipe &lt; SYS1.PARMLIB(PROG00)</td>
<td>RxPipe &lt; SYS1.PARMLIB(PROG00)</td>
</tr>
<tr>
<td></td>
<td>fanout</td>
</tr>
<tr>
<td></td>
<td>:1 w1 w3 w4 where word(r,1)=&quot;APF&quot; &amp; pos('(DATA02)',r)&gt;0</td>
</tr>
<tr>
<td></td>
<td>:2 w1 w3 w4 where word(r,1)=&quot;LNKLST&quot; &amp; pos('ADCD',r)&gt;0</td>
</tr>
<tr>
<td></td>
<td>:3 where \catch</td>
</tr>
<tr>
<td></td>
<td>load 1 clear</td>
</tr>
<tr>
<td></td>
<td>load 2</td>
</tr>
<tr>
<td></td>
<td>fanout</td>
</tr>
<tr>
<td></td>
<td>:1 where word(r,1)=&quot;APF&quot; &amp; pos('(DATA02)',r)&gt;0</td>
</tr>
<tr>
<td></td>
<td>:2 where word(r,1)=&quot;LNKLST&quot; &amp; pos('ADCD',r)&gt;0</td>
</tr>
<tr>
<td></td>
<td>fanin</td>
</tr>
<tr>
<td></td>
<td>:1 w1 w3 w4</td>
</tr>
<tr>
<td></td>
<td>:2 w1 w3 w4</td>
</tr>
</tbody>
</table>

**Notes**
- The **FANOUTs** also reformat the saved records.
- The **FANOUT** for .3 is not really necessary.
- **FANIN** will clear the stack.
- The reformat is done by the **FANIN**.

The 2 programs do the same, just using different methods.

**FANOUT**

Save stack to multiple internal streams by filter.

Any number of streams can be defined, each definition must be preceded by a colon. The stack records are parsed to variables R (entire record) and W1-W10 (words 1 to 10) before the test is done. If all tests fail, then the record is ignored. Records may be written to multiple streams.

The value ‘!?CATCH’ is set if a record is written to a stream, so you can use the expression WHERE \!CATCH to write records not caught by previous filters.

**FANIN**

Load internal streams to the stack, applying filters and formats. Similar to the LOAD command, except stack is cleared, filter and format can be applied individually to each stream and streams are loaded sequentially.

Any number of streams can be defined, each definition must be preceded by a colon. The stream records are parsed to variables R (entire record) and W1-W10 (words 1 to 10) before the test is done. If all tests fail, then the record is ignored. Default is no filter.
Looping over the stack – FOREACH

**FOREACH – loop over stack**

Program to list member contents

You can use each stack record to drive a RXPIPE command.

The demo lists SYS1.PARMLIB members starting with IGD, putting a header line in front of each.

Note that the stack is not preserved between command invocations, so avoid commands that modify existing stack entries.

The sub-command delimiter (:) can be changed by the DLM parameter, which must be first.

The records are parsed when they are pulled from the stack, so you can use the variables &R and &W1 - &W20 in the commands, plus internal variables and variables generated by the called commands.

The ‘strip’ command makes the record eligible for the include by mask. You could use the command ‘mbrlist sys1.parmlib(igd*)’ instead of listds + drop + strip, mbrlist is a separate command member, though still part of the package.
Using external REXX function

The sample uses the external REXX function RXVSAMBA, which can read from and write to a VSAM cluster directly.

Notes
- The command must be a REXX function, thus use 'call' or cc.
- $result may or may not contain proper value.
- The SAY command shows some statistics and potential error message.
- The samples also show the IF .. ELSE construct.

The sample tests the return value from the program.

The RXVASAMBA function can write data to the stack, which is then displayed automatically by RXPIPE at the end.

Using external REXX function (2)

The samples use the external REXX function REXXGBLV to pass a stem between RXPIPE and the caller.

REXXGBLV can save a REXX stem or stack, and later reload as stem or stack.
Some Internals

The first program
RxPipe lstsc ent('DVL.PP.RXPIPE.V007.LIB') vol
| exec ACCOUNT-INFO
| split ' ' 
| strip b

Is converted to
Interpret ,
Call 00EXEC "lstsc ent('DVL.PP.RXPIPE.V007.LIB') vol";
Call 00EXEC "exec ACCOUNT-INFO";
Call 00EXEC "list before split";
Call 00EXEC "split ' '";
Call 00EXEC "strip b"

RXPIPE command NDUP
ON Dup: Procedure expose $stacked ($globalv) /* Drop duplicate lines */
dc = 0
  do $stacked 
    parse pull r 
    if dc.r then iterate 
    dc.r=1 
    queue r 
  end 
return 0

Entire blocks are INTERPRETed, not single lines, i.e.
Interpret “Do queued(); parse pull r; -do something--; end”

Notes
• 00EXEC determine type of command and executes accordingly. It also terminates processing if error condition is encountered, or return code is greater than limit.
• Commands starts with a zero, they may be procedures.

The RXPIPE parameter is converted to a REXX interpret-able program, as that allows for much simpler IF-ELSE conditional handling.

Commands are internally named with a leading zero, to avoid conflict with external commands or programs. RXPIPE does an initial scan of itself to identify those internal commands.

There is an active ON SYNTAX. Neither ON ERROR nor ON NOVALUE are implemented as they have shown to be disruptive here. I highly recommend ON NOVALUE in general.

Most hi-use functionality is internal to RXPIPE, this includes parameter parsing and mask handling.

Sorting is done using an external member, which uses the ‘quick’ sort technique. Other sorts like USS and DF/SORT are supplied and can be used instead.

A number of the internal commands are processed using the REXX INTERPRET command. They are somewhat optimized in that entire blocks are INTERPRETed, not single lines, i.e.
  Interpret “Do queued(); parse pull r; -do something--; end”
Other bits and pieces

**Built-in variables**

- **$DATE**: Today's date in expanded European format yyyy-mm-dd
- **$HALTONRC**: If set, then halt if $rc is ge the value. Set the value using the SET command, i.e. SET $HALTONRC = 8. Default is ‘not set’.
- **$JOBNAME**: The jobname/TSOid RXPIPE is running in
- **$LASTDD**: Latest DD name generated by allocation
- **$LASTDS**: Latest Dataset name generated by allocation
- **$MSGLVL**: Show commands and their result if $MSGLVL is set gt 0.
- **$RC**: Latest numeric return value
- **$RESULT**: Latest return value, if any
- **$TIME**: Time in 24-hour format with seconds hh:mm:ss
- **$USER**: Userid
- **$USERPFX**: The TSO PROFILE PREFIX value
- **JOBNAME**: Same as $JOBNAME
- **MCATNAME**: Mastercatalog
- **MCATVOL**: Mastercatalog volume
- **ME**: Same as $USER
- **R**: The current stack record for some commands
- **USER**: Same as $USER
- **USERPFX**: Same as $USERPFX
- **W1-W20**: Word 1 to 20 of the record for some commands

Some commands produce variables of their own.
Thank you for listening

Comments?

Questions?