#SHAREorg

SHARE
Technology · Connections · Results

CMS Pipelines Hands-On Workshop

Rick Barlow
Nationwide Insurance

March 12, 2012
10442

Agenda

- Introduction
- What are Pipelines?
- PIPE syntax
- Pipeline stages
- Simple pipes
- Hands-on
- Categories of stages
- Reference Information
- Contact Information
- Acknowledgements
- Pipes in REXX
- Appendices
  (lists of Pipeline stages)
- REXX Interface sample programs
  DUMPVARS
  DUMPREXX

Introduction

- This session is intended as an INTRODUCTION to using the CMS PIPE command as a part of programming an application under CMS using REXX.
- A working knowledge of coding in REXX is assumed.
- You should not expect to leave understanding everything there is to know about the CMS PIPE command. You should understand the basic concept of PIPEs.
- Significant documentation of the various PIPE stages is available through the VM HELP facility and through the authors PIPE AHELP stage.

What Are Pipelines?

- The Pipeline Concept
  - A pipeline is simply a series of programs through which data flows, just as water flows through the sections of a water pipe. In a pipeline, a complex task is performed by processing data through several simple programs in an appropriate sequence.

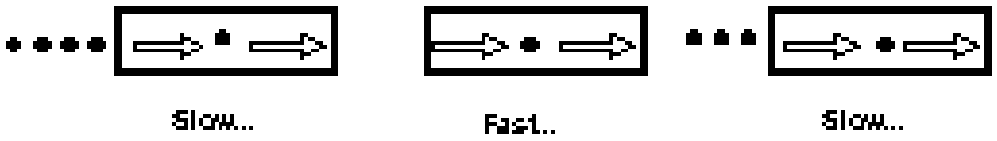
- A word of caution:
  - It is important to remember that, in most cases, all records placed into a pipeline must flow all the way through. This can dramatically affect the performance of a pipeline which reads a large file to select only a portion of the records.

What Are Pipelines?

- CMS Pipelines are made of programs called “Stages”
- Each stage:
  - Reads data from the pipeline
  - Processes (optionally transforms) it in some way
  - Writes the (transformed) data back to the pipeline
- Data is automatically presented as input to the next stage
- Individual stages are totally independent of one another
  - They do not need to know what comes before or after.
- They are also device independent
  - read from or write to any input or output device.

What Are Pipelines?

- 'Pipe-Think'
  - Pipe-Think is a term that has been created to help conceptualize that it is important to think as a plumber when attempting to code pipelines. The flow of data through a pipeline does not stop at each stage. The first data into the Pipe passes to the next stage as soon as the stage has completed its function. The effect is that data can be processed in multiple stages at the same time.



PIPE Syntax

- The PIPE command format:
  PIPE pipeline-specification (pipe options)
  - The argument to PIPE is a "pipeline specification". A pipeline specification is a string of characters listing the stages to be executed.

- PIPE stage-1 | stage-2 | stage-3 | stage-4 | ... | stage-n

PIPE Syntax

- The stage separator
  - Separates the stages in pipeline specification
  - Default is vertical bar (“|”) - x'4F‘
    - PC keyboard -> Shift-Backslash
  - Change with the STAGESEP or SEParator option on the PIPE command
    - If you want to use exclamation point (!) for a stage separator, a PIPE command might look like this:

      PIPE (STAGESEP !) stage-1 ! stage-2 ! ... ! stage-n

PIPE Syntax

- The end character:
  - Used to concatenate multiple pipeline programs into a single pipeline specification
  - Necessary for stages which can accept multiple input streams or write multiple output streams
  - A frequently used end character is tilde (“~”) (Shift-back-quote on the keyboard)
  - Specify using the ENDchar or ESCape option on the PIPE command
    - PIPE (ENDCHAR ~) stage-1a | stage-1b ~ stage-2a | stage-2b

Pipeline Stages

- The stages which come with CMS Pipelines are divided into three main categories
  - Device Drivers
    - interaction between the pipeline and the environment where it runs
    - The majority of the current Device Driver stages are listed in Appendix One.
  - Filters
    - provide the ability to modify the contents of the stream of data flowing through the pipeline by various types of selection, conversion and modification
    - The majority of the current Filter stages are listed in Appendix Two.

Pipeline Stages - continued

- Other stages
  - Control
  - Gateways which provide for branching within the pipeline
  - Host Command Processors
  - Others miscellaneous stages
  - The majority of the current Other stages are listed in Appendix Three.

Pipeline Construction

- The correct term for input to and output from Pipeline Stages is a stream
- Most stages can accept one or more input streams
- Most stages can generate one or more output streams
- Some Device Driver stages only handle input or output
- Connections for multiple streams are coded in a Pipeline specification using labels
  - A label is a string of up to 8 alphanumeric characters followed by a colon
  - First occurrence is the label definition
  - Subsequent use is called a label reference

Let's get started – Device Driver Stages

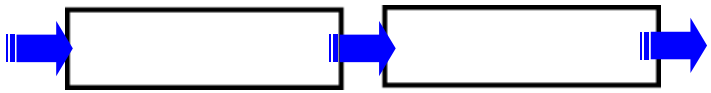
- A simple pipe may contain only device drivers. Here is a traditional one to start with:

```
PIPE LITERAL Hello, World! | CONSOLE
```

- Another simple pipe performs an echo operation:

```
PIPE CONSOLE | CONSOLE
```

  This pipeline continues reading from the console and writing back until it reaches end-of-file, i.e., until it receives a null line as input.



  Try these

Simple PIPEs

- All pipeline stages perform their designated function as well as passing all of the date through to subsequent stages.
- Consider this example:
  - **`PIPE CONSOLE | CONSOLE | CONSOLE`**
- What will happen?

```
pipe console | console | console
Test1
Test1
Test1
Test2
Test2
Test2
```

Simple PIPEs

- Simple pipelines using only device drivers could be executed to copy files from any valid device and/or file to any other device or file.
  - **PIPE < PROFILE EXEC A | > PROFILE COPY A**
  - **PIPE < PROFILE EXEC A | CONSOLE**
  - **PIPE CONSOLE | FILEFAST NEW FILE A**
  - ```
    'PIPE FILEFAST' fn ft fm '|',
      '>' outfn outft outfm '|',
      'CONSOLE |'
      'PUNCH'
    ```


    ```

Filters

- Pipelines built only of device drivers do not really show the power of CMS Pipelines. There are dozens of built-in programs included with CMS Pipelines. Most of them are "filters"; programs that can be put into a pipeline to perform some transformation on the data flowing through the pipeline.
- A simple pipeline consisting of a couple of device drivers wrapped around a few filter stages can provide an instant enhancement to the CMS command set. Once you have some practice, you may find yourself typing lots of little "throwaway" pipes right on the command line.

Filters

Buffering

BUFFER	Collect all records in a stage before passing any
ELASTIC	Buffer only enough to prevent stall of following stages
SORT	Arrange records in ascending or descending order

Cut and Paste

SPLIT	Split records into multiple records
STRIP	Remove leading and/or trailing characters from records

Record Format

SCM	Line up comments and complete unclosed comments in REXX or C programs
SPECS	Rearranges the contents of records

Filters

Selection Filters

DROP	Discards one or more records
FIND	Select records that begin with a specified text
LOCATE	Select records that contain a specified string of characters
NFIND	Select records that do not begin with a specified text
NLOCATE	Select records that do *not* contain a specified string
TAKE	Select one or more records from the beginning or end of the primary input stream
UNIQUE	Compare the contents of adjacent records and discards or retains the duplicate records

Other Filters

COUNT	Count bytes, blank-delimited character strings, records, or the length of the longest or shortest line

Filters

- Most of the filter programs are self-explanatory
- Often behave like the XEDIT subcommand of the same name
- Records that match the criteria in a filter stage are passed to the primary output stream – the next stage in the Pipeline
- If a label is coded on the filter stage, records that do not match the criteria are passed to the secondary output stream if it is connected

Filters

- This pipeline will locate a user-id within the response of the CP QUERY NAMES command:

```
PIPE CP QUERY NAMES | LOCATE /RSCS/ | CONSOLE
ESAWEB01 - DSC , ESAADMIN - DSC , RSCSDNS2 - DSC , RSCSDNS1 – DSC
PAGEMAN  - DSC , TCPIP    - DSC , RSCSLOG  - DSC , RSCS3    - DSC
RSCS2    - DSC , RSCS     - DSC , HIDRO    - DSC , VMARCH   - DSC
```

- Notice that you get an entire line displayed. You could split the response at commas and remove leading blanks with:

```
PIPE CP QUERY NAMES | SPLIT AT , | STRIP | LOCATE /RSCS/ | CONSOLE
RSCSDNS2 – DSC
RSCSDNS1 – DSC
RSCSLOG  - DSC
RSCS3    - DSC
RSCS2    - DSC
RSCS     - DSC
```

Filters

- Another filter much like LOCATE is FIND. These filters behave much like their counterparts in the XEDIT environment.
- The same two pipelines we just looked at with FIND in place of LOCATE:

```
PIPE CP QUERY NAMES | FIND RSCS | CONSOLE
RSCS2    - DSC , RSCS     - DSC , HIDRO    - DSC , VMARCH   - DSC
PIPE CP QUERY NAMES | SPLIT AT , | FIND RSCS | CONSOLE
RSCS2    - DSC
```

- We expected to find all occurrences of RSCS. What happened?
  - The FIND filter always looks for the data starting in column 1.
- Fix the previous Pipeline by adding a STRIP stage.

Filters

- The FIND filter also does not use a delimiter to determine the start and end of the search string. Therefore, it uses all of the characters until it finds a space or stage separator.
- If we only want to find the single user-id RSCS, we need to use an underscore to signify that a blank should follow RSCS. This is the same syntax as the XEDIT FIND subcommand.

```
PIPE CP QUERY NAMES | SPLIT AT , | STRIP | FIND RSCS_ | CONSOLE
RSCS     - DSC
```

Filters – NOT

Reverse Selection - Not “Don't use them”

- There are filters to select all records that do **not** match the criteria
- If you intend to process both the records that do and do not meet the criteria, you can use either the positive or negative version of the selection stage with a label
- NLOCATE is the pipeline stage to exclude records that include a given string anywhere in the record.

```
PIPE CP QUERY NAMES | SPLIT AT , | NLOCATE /DSC/ | TAKE 5 |
CONSOLE
```

- NFIND is the pipeline stage to exclude records which begin with a string.

```
PIPE CMS QUERY ACCESSED | NFIND Mode | CONSOLE
```

Filters

- Another useful filter is COUNT. It can return various characteristics about the data in your pipeline including the number of lines or words.
- If you want to know how many records are in a file, you could type:

```
PIPE < CPQ NAMES | COUNT LINES | CONSOLE
```

- If you want to know how many words are in a file, you could type:

```
PIPE < CPQ NAMES | COUNT WORDS | CONSOLE
```

- You could even determine the number of unique words with:

```
PIPE < CPQ NAMES | SPLIT | SORT UNIQUE | COUNT WORDS | CONSOLE
```

Filters

- TAKE [FIRST|LAST] [n]
  - Select 1 or more records from one end of the stream of data
  - FIRST - select from the beginning
  - LAST – select from the end
  - [n] is the optional number of records – default is 1
- DROP [FIRST|LAST] [n]
  - Omit 1 or more records from one end of the stream of data
  - FIRST - select from the beginning
  - LAST – select from the end
  - [n] is the optional number of records – default is 1

Filters

- SORT
  - Re-order the records in the stream
  - This stage will cause all records to be delayed (buffered)
  - Sort by word(s) and/or column(s)
  - Combine with other stages for specific results
- UNIQUE
  - Use to cause a single copy of a given record / word / columns to pass to the primary output stream
  - FIRST – keep the first record with the selected location
  - LAST – keep the last record; this is the default

Filters

- Using device drivers and filters can be very useful for writing “throw away” Pipelines at the CMS command line.
- Filters become most useful when used in combination.
- Suppose you wanted to see the last disk that was accessed in your virtual machine. You could use the SORT and TAKE stages like this:

```
PIPE CMS QUERY ACCESSED | NLOCATE 1.4 /Mode/ | SORT 1.1 D | TAKE
| CONSOLE
```

- Even more simply

```
PIPE CMS QUERY ACCESSED | NFIND Mode | SORT W1 D | TAKE | CONSOLE
```

Other Stages

Control Stages

APPEND	Write primary input stream records to the primary output stream followed by records from a specified device driver

Gateways (pipe fittings)

FANOUT	Copy primary input stream records to multiple output streams
FANIN	Combine multiple input streams into a single stream in a specified order
FANINANY	Combine multiple input streams into a single stream. FANINANY reads an input record from any input stream that has a record available.
GATHER	Read records from all connected streams in sequential order or other specified order

Host Command Processors

CMS	Issue CMS commands with full command resolution
COMMAND	Issue CMS commands as if they were invoked using **Address 'COMMAND'** from REXX/VM
CP	Issue CP commands

Host Command Stages

- Issue host commands and routes the responses into the pipeline.
  - CP issues CP commands
  - COMMAND issues CMS commands using program call
    - Parsing behaves like REXX Address 'COMMAND'
    - Uses extended parameter list
  - CMS issues CMS commands with full command resolution
    - Parsing behaves like the CMS command line or REXX Address 'CMS'
    - Uses CMS subcommand environment
- Each of these stages issues its argument string as a command and then reads the input stream and issues each record as a commands. The command responses are captured and each line becomes a record in the pipeline.

```
PIPE LITERAL QUERY TIME | CP QUERY USERID | CONSOLE
BARLOWR  AT VMB
TIME IS 21:55:24 EST WEDNESDAY 2012-02-15
CONNECT= hh:mm:ss VIRTCPU= mmm:ss.hh TOTCPU= mmm:ss.hh
```

- The return code of each command is written to the secondary output stream

Host Command Stages

- A simple pipeline to talk to CMS might look like this:

```
PIPE LITERAL QUERY ACCESSED | CMS IDENTIFY | CONSOLE
```

If you place it in an EXEC, it might look like this:

```
/* Who am I and what is my environment like */
'PIPE(NAME CLASS)|',
   'LITERAL QUERY ACCESSED |',
   'CMS IDENTIFY |',
   'CONSOLE'
```

```
BARLOWR  AT VMB      VIA RSCS     2012-02-20 23:09:51 EST      MONDAY
Mode  Stat     Files  Vdev  Label/Directory
A      R/W      2299  191   RB191B
S      R/O       709  190   MNT190
Y/S    R/O      1232  19E   MNT19E
```

More Built-in Programs: SPECS

- One of the most complex and least obvious stages
- Simple form: selects pieces of an input record and puts them into the output record
- Basic syntax is taken from the SPECS option of the CMS COPYFILE
- Includes many more features

  SPECS input-location1 output-location1 ...
- Specify as many input/output pairs as you need

More Built-in Programs: SPECS

- Input location options
  - Location
    - start-end — e.g. 1-* or 10-14
    - start.length — e.g. 10.5
  - wordrange or wnumberrange
    - select blank-delimited word(s)
    - examples — WORD 2.5 Word 8 W -2;-1
  - literal
    - a delimited string — e.g. /Some Thing/
  - RECNO
    - the current records number in the stream

More Built-in Programs: SPECS

- Output location options
  All output location options may include a length
  - Explicit location
    - starting column number — e.g. 10
    - start**-**end — e.g. 10-14
    - start**.**length — e.g. 10.5
  - Relative location
    - **N**ext column of the record — e.g. NEXT or N.8
    - **N**ext**W**ord – leaves one space — e.g. NEXTWORD or NW.4

More Built-in Programs: SPECS

- More complex features include
  - numeric calculations
  - field summarization
  - report generation
- SPECS documentation
  - Chapter 14 of the CMS/TSO Pipelines: Author's Edition
  - PIPE AHELP SPECTUT
  - http://vm.marist.edu/~pipeline/ahelp/spectuto.html

More Built-in Programs: SPECS

- Examples

```
PIPE LITERAL Hello, there.|SPECS W1 1 W-1;-1 15|CONSOLE
Hello,          there.

PIPE LITERAL Hello, there.|SPECS 1-5 1.10 6 NW 8-* 20|CONS
Hello       ,         there.
```

More Built-in Programs: SPECS

- Conversion routine
  - Add between input and output location
  - Similar to REXX conversion function
    - C2B, C2D, C2X, D2C, X2C…
  - Applied to input field before moving to output field
- Example

```
PIPE LITERAL C885939396|SPECS 1-* X2C 1|CONSOLE
Hello
```

More Built-in Programs: SPECS

- Placement option
  - Add after output location
  - Cause alignment in the output location
    - "LEFT", "CENTER" or "RIGHT"
- Example

```
PIPE LITERAL Record Number|
   SPECS W1 1 W2 NW.10 RIGHT RECNO 46.4 RIGHT|
   CONSOLE
Record     Number                              0001
```

More Built-in Programs: SPECS

- Suppose you want to take the list of disks we created before and show only the virtual address and access mode. You could use the following pipeline:

```
PIPE COMMAND QUERY ACCESSED | SPECS W1 1 W4 NW | CONSOLE
Mode Vdev
A 191
S 190
X DIR
Y/S 19E
```

- Perhaps you would like to get the last 2 words of a record. To do that, you use the 'wnumberrange' format like this:

```
PIPE COMMAND QUERY ACCESSED | SPECS WORD -2;-1 1 | CONSOLE
Vdev  Label/Directory
191   RB191B
190   MNT190
DIR   VMBSYS:BARLOWR.VM.STUFF
19E   MNT19E
```

SPECS exercise

1. Find out what disks and/or directories are accessed
   - Hint: Use COMMAND QUERY ACCESSED
2. Modify the Pipeline to show just the access mode, VDEV or DIR and the Label/Directory
3. Modify the Pipeline to line up the columns
4. Modify the Pipeline to number the lines of output

Host Command - Build a Host Command

- A simple REXX EXEC to talk to CP might look like this:

```
PIPE CP Q USERID|CONSOLE|SPECS /Q/ 1 W1 NW|CP|CONSOLE

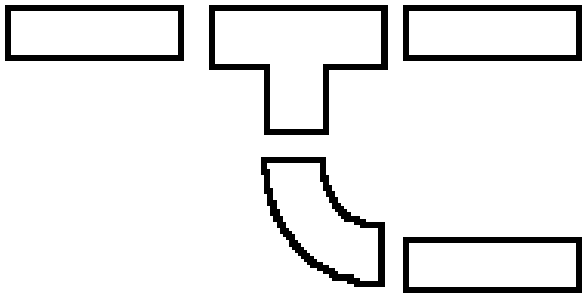
/* Who am I and where am I logged on */
'PIPE(NAME CLASS)|',
   'CP QUERY USERID |',
   'CONSOLE |',
   'SPECS /QUERY/ 1 Word 1 NextWord |',
   'CP |',
   'CONSOLE'
```

- The resulting output would resemble this:

```
BARLOWR  AT VMB
BARLOWR  -L00E4
```

Gateway Stages

- The majority of the Gateway stages are similar to the fittings that you might include in a water pipe. You might look at them like a 'T'. The most common are the FANOUT, FANIN, FANINANY and GATHER stages.
- When a record in the stream is rejected by a selection filter, the record is placed on the secondary stream of that stage. Labels are used to process the records in an additional pipeline.



Gateway Stages

- For example, we could assemble a pipeline where we send the CMS Query disk output into two streams like this:

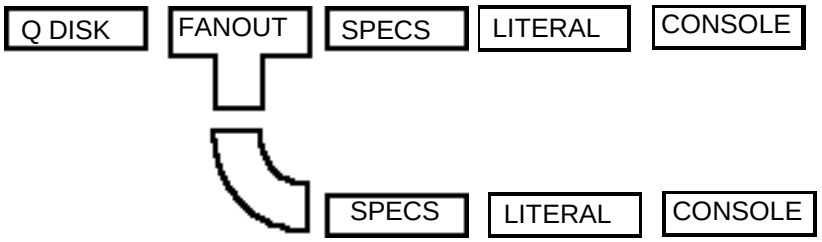
```
/* */
'PIPE(ENDCHAR ?) |',
   'CMS QUERY DISK |',
   'DROP |',
'F: FANOUT |',
   'SPECS /1/ 1 1-69 NW |',
   'LITERAL This stuff is in the first stream: |',
   'CONSOLE',
'?',
'F: |',
   'SPECS /2/ 1 1-69 NW |',
   'LITERAL This stuff is in the second stream: |',
   'CONSOLE'
```

Gateway Stages

- Result:

```
This stuff is in the first stream:
This stuff is in the second stream:
1 A      R/W      2299  191   RB191B
2 A      R/W      2299  191   RB191B
1 S      R/O       709  190   MNT190
2 S      R/O       709  190   MNT190
1 Y/S    R/O      1232  19E   MNT19E
2 Y/S    R/O      1232  19E   MNT19E
```

- What is wrong?



Gateway Stages

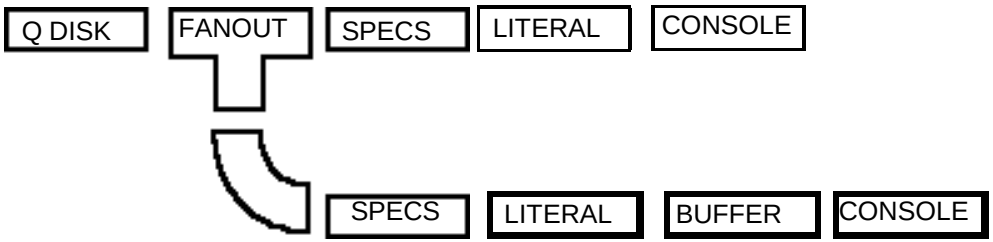
- To get the output in the order we expect, one thing we can do is to delay the data in the second pipeline:

```
/* */
'PIPE(ENDCHAR ?) |',
   'CMS QUERY ACCESSED |',
   'DROP |',
'F: FANOUT |',
   'SPECS /1/ 1 1-69 NW |',
   'LITERAL This stuff is in the first stream: |',
   'CONSOLE',
'?',
'F: |',
   'SPECS /2/ 1 1-69 NW |',
   'LITERAL This stuff is in the second stream: |',
   'BUFFER',
   'CONSOLE'
```

Gateway Stages

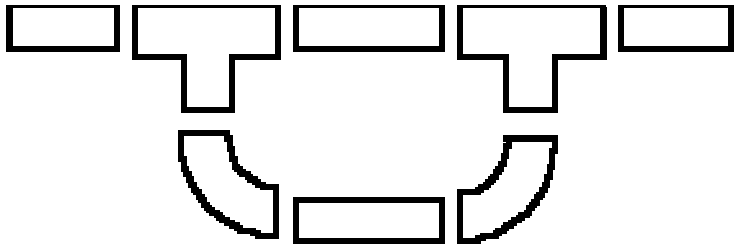
- Result:

```
This stuff is in the first stream:
1 A      R/W      2299  191   RB191B
1 S      R/O       709  190   MNT190
1 Y/S    R/O      1232  19E   MNT19E
This stuff is in the second stream:
2 A      R/W      2299  191   RB191B
2 S      R/O       709  190   MNT190
2 Y/S    R/O      1232  19E   MNT19E
```



Gateway Stages

- To eliminate two CONSOLE stages, we could use a FANIN stage.



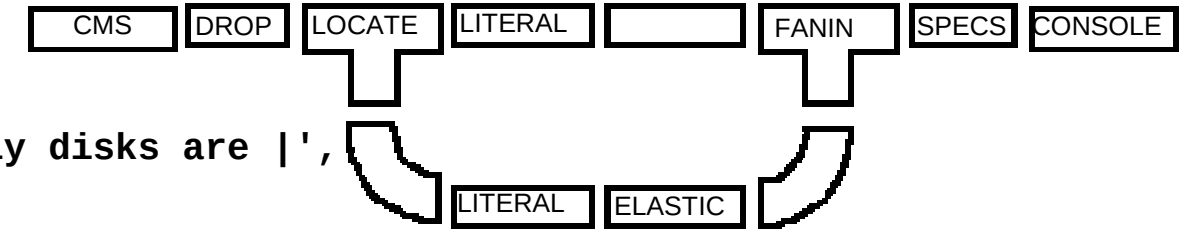
```
/* */
'PIPE(ENDCHAR ?) |',
   'CMS QUERY ACCESSED |',
   'DROP |',
'F: FANOUT |',
   'SPECS /1/ 1 1-69 NW |',
   'LITERAL This stuff is in the first stream: |',
'I: FANIN |',
   'CONSOLE',
'?',
'F: |',
   'SPECS /2/ 1 1-69 NW |',
   'LITERAL This stuff is in the second stream: |',
   'ELASTIC',
   'I:'
```

Gateway Stages

Another way that the data can be split into multiple streams is through selection filters. Data that is selected goes to the primary output stream. Unselected data passes to the secondary output stream. For example:

```
/* */
Address 'COMMAND'
'PIPE(ENDCHAR ?) |',
   'CMS QUERY ACCESSED |',
   'DROP |',
'L: LOCATE ,R/W, |',
   'LITERAL The Read/Write disks are |',
'F: FANIN |',
   'SPECS W1.4 1 |',
   'CONSOLE',
'?',
'L: |',
   'LITERAL The Read Only disks are |',
   'ELASTIC |',
   'F:'
```

```
The Read/Write disks are
A       R/W       2299  191
The Read Only disks
S       R/O        709  190
Y/S     R/O       1232  19E
```



Daily Use

- "Throw away" – Pipelines used at the command line
  - The bigger your system(s), the more useful
    - Filter responses from CP commands
      - *Query NAMES – select the ones you want to see*
    - Use selections from CP commands to build and run more commands
      - *Query* devices – *select the ones you want; generate more CP commands; execute; filter the responses*
- Pipelines in REXX for repeated use

Exercises

- Massaging CP command output
  - Use Query Virtual DASD
    - CP QUERY VIRTUAL DASD
  - (a) Show on the console
  - (b) Use the output from the CP command to build commands to issue QUERY MDISK DETAILS for each device
  - (c) Improve readability
    - Insert blank lines

Exercises

- Issue CP command and filter the results
  - Because this is a lab and user-ids have no privileges, read from a file
    - Instead of **`CP Query DASD ALL`**
    - Use **`< CPQ DASDALL`**
  - (a) Show the first 10 on the console
  - (b) Show the first 10 and the last 10
    - This will need a label and two streams
- It is important to know the data you are processing
- It often requires some experimentation to get the results to look as you intend
  - (c) Show the first and last 10 after: removing OFFLINE and BOXED
  - (d) Show the first and last 10 after: removing OFFLINE, BOXED and devices with no label

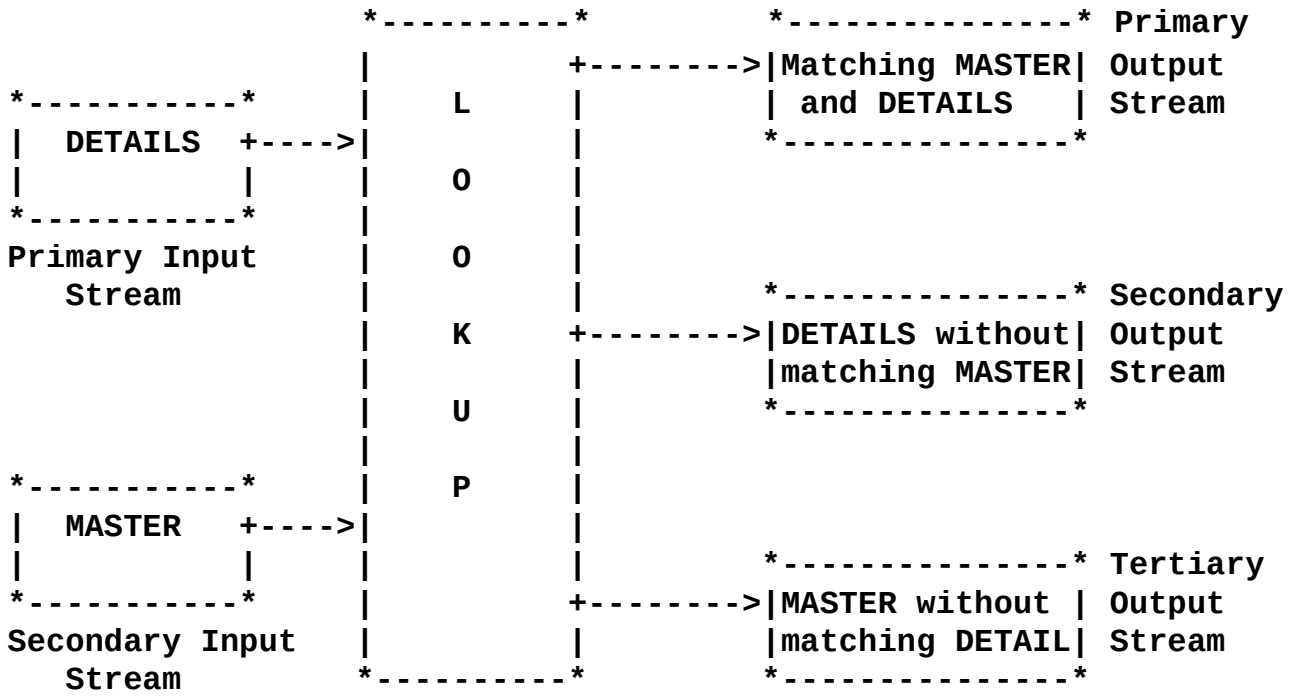
Exercises

- Build a list of lab user-ids and find out if they are logged on
  - Userids are PIPUSR00 through PIPUSR30
  - Query if they are logged on
  - Show the results and Return Codes
    - Reminder: The secondary stream from the CP stage has the Return Codes

- There are multiple ways to solve this
- The answers include a very simple solution as well as a solution with less obvious stages

If Time Permits…

- LOOKUP – Find records in a reference
- Both Master and Detail records need to be sorted the same



If Time Permits…

```
/* MYLOOKUP EXEC */
'pipe (endchar ?) |',
  '< DETAIL RECORDS |',       /* read detail records */
'l: lookup |',                /* find matches */
  '> MATCHING RECORDS A |',   /* write matching masters and details */
'?',                          /* start of second pipeline */
  '< MASTER RECORDS |',       /* read master records */
'l: |',                       /* define secondary streams for LOOKUP */
  '> UNREF DETAILS A |',      /* write details without masters */
'?',                          /* start of third pipeline */
'l: |',                       /* define tertiary output for LOOKUP */
  '> UNREF MASTERS A'         /* write masters without details */
Exit rc
```

If Time Permits…

- LOOKUP
  - Use CPQ DASDALL file as the master
    - Discard the OFFLINE and BOXED devices
  - Use SELECT LIST file as the details

Reference

- **HELP PIPE MENU**
- PIPE AHELP MENU
- SC24-6169.....CMS Pipelines Reference
- CMS/TSO Pipelines Runtime Library Distribution web site http://vm.marist.edu/~pipeline

Contact Information

Rick Barlow

Senior z/VM Systems Programmer

Phone: (614) 249-5213

Internet: Richard.Barlow@nationwide.com

Acknowledgements

- CMS Pipelines is the work of John Hartmann of IBM Denmark.

- Much of the information has been gathered from the product and presentations at SHARE by John and Melinda Varian of Princeton University. Melinda was a long time VMer and ***the*** Pipelines advocate in the VM community from before Pipelines was integrated into the VM product. She did an outstanding job of spearheading the acceptance of Pipelines and helped to drive the effort for continuing development and inclusion in the CMS product.

PIPEs in REXX

- If you want to include a PIPE command in a REXX EXEC, keep in mind that the entire command is a string, only portions of which should have variables substituted. For example, the last PIPE would look like this in a REXX EXEC:

```
'PIPE <' fn ft fm '| >' outfn outft outfm '| CONSOLE | PUNCH'
```

  You should quote the parts that are not variable while allowing REXX to substitute the correct values for the variable fields.

PIPEs in REXX – XEDIT Macros

- A PIPELINE written as a single continuous string is known as LINEAR format.
- XEDIT macros have a maximum string length of 255 characters. Therefore, when including complex Pipelines in XEDIT macros it may be necessary to include Address 'COMMAND' on the PIPE command to avoid truncating the pipeline specification.
- Putting literals in quotes is important so that later coding does not change the program by using a word that you used as a literal for a variable.
- For example, you could include the statement
  `Address Command`
  in a REXX program. If a future change proceeding this statement in the program assigns command = 'ERASE', your statement will evaluate to Address 'ERASE' and that will generate an error.

REXX Coding Sideline

- Choose a coding style and stick to it
  - Improves readability when coding is consistent
  - Understand what is a command, literal, etc
- Things to consider
  - Case: all upper, all lower, mixed - when?
  - Indentations
  - Comments
  - Spaces around operators (= + - |)

REXX Coding Sideline

- My choices
  - REXX commands Capitalized
  - Pipelines
    - Stages Upper case
    - Separator to the right
  - Literals in single quotes in desired case
  - Variables all lower case
  - Indentations
    - 2 characters
    - End is indented with the rest of the group it ends

Appendix One: Device Drivers

CMS files

• BFS	• Connect to a byte stream file in a byte file system BFS can be either a read or write stage.
• BFSDIRectory	• Read the contents of an existing byte file system directory
• BFSQuery	• Get info from OpenEdition about current working directory
• BFSREPlace	• Write records from primary input stream to replace a byte stream file
• BFSSTATE	• Retrieve status information about byte stream files
• BFSEXECUT	• Read records from primary input stream and send each request to OpenEdition services

Appendix One: Device Drivers

CMS Libraries

• LISTPDS	• Read the directory of a CMS simulated PDS (e.g. MACLIB)
• MACLIB	• Generate a macro library from COPY file members
• MEMBERS	• Extract members from a MACLIB, TXTLIB or a file of similar format
• PDSDIRECT	• Write directory information from a CMS simulated PDS

REXX

• REXXVARS	• Retrieve information about REXX variables
• STEM	• Get or set REXX variables with the specified stem
• VAR	• Get or set a REXX variable
• VARLOAD	• Set a REXX variable

Appendix One: Device Drivers

Terminal

- CONSOLE — Read or write the terminal in line mode
- FULLSCR — Full-screen 3270 write and read to the console or attached 3270 device
- MESSAGE — Issue a message from a Pipeline
- 3270BFRA — Convert a 2-byte unsigned integer to the 12-bit buffer address required for some 3270 devices, or vice versa
- 3270ENC — Prepares a 64-character translate table used to convert binary values to displayable 1-byte graphic characters

Unit Record

- PRINTMC — Print lines with machine carriage control characters
- PUNCH — Punch cards
- READER — Read from a virtual card reader
- URO — Write records to a virtual printer or virtual punch

XEDIT

- XEDIT — Read or write an XEDIT in-storage file
- XMSG — Issue XEDIT messages

Note: Use XEDIT stage in XEDIT macros. It is much more efficient and better performing than STACK.

Appendix One: Device Drivers

Other Device Drivers

• BEGOUTput	• Use in REXX extensions to indicate anything directed to the subcommand environment is written to the selected output stream
• DELAY	• Suspend stream until a specific time or interval has passed
• EMSG	• Issue messages
• HOLE	• Discard records
• IMMCMD	• Write the argument string from immediate commands
• ISPF	• Access ISPF tables
• LITERAL	• Write argument string to the output stream before copying input stream to output stream
• MDISKBLK	• Read blocks of data from an accessed CMS minidisk
• OUTPUT	• Write a record to the currently selected output stream
• PEEKTO	• Inspect the next record in the selected input stream without consuming the record
• READTO	• Read the next record from the currently selected input stream

Appendix One: Device Drivers

Other Device Drivers

• QSAM	• Read or write sequential datasets through a DCB
• SETRC	• Set return code for an output subcommand
• SQL	• Interface to SQL
• SQLCODES	• Write the last 11 SQL codes received
• STACK	• Read or write the program stack
• STORAGE	• Read or write virtual machine storage
• STRLITERAL	• Write argument string to the output stream before copying input stream to output stream
• TAPE	• Read or write tapes at the current position
• VMC	• Write VMCF reply
• XAB	• Read or write External Attribute Buffers (XABs)
• XRANGE	• Create on record containing a specified range of characters

Note: Avoid STACK if possible.

Appendix Two: Filters

Assembler Files

- ASMCONT
  - Join multiline Assembler statements
- ASMXPND
  - Split Assembler statements

Buffering

- BUFFER
  - Collect all records in a stage before passing any
    Note: Avoid BUFFER if possible - it slows down execution.
- COMBINE
  - Combine records into one record using a logical operator
- COPY
  - Delay by one record the passing of records from the input stream to the output stream to prevent a pipeline stall
- ELASTIC
  - Put a sufficient number of input records into a buffer to prevent a pipeline stall
- INSTORE
  - Reads records from its input stream into storage and writes a single record containing only the pointers to the records in storage.
- OUTSTORE
  - Unload a file loaded into storage by INSTORE
- SORT
  - Arrange records in ascending or descending order

Appendix Two: Filters

“Cut and Paste”

• CHOP	• Selectively truncate records
• JOIN	• Concatenate groups of records
• PAD	• Extend records to a specified length
• SPLIT	• Split records into multiple records
• STRIP	• Remove leading or trailing characters from records
• JOINCONT	• Join records marked with a continuation string

Appendix Two: Filters

Format Conversion

BLOCK	Block records
BUILDSCR	Build a 3270 data stream
DEBLOCK	Deblock external data formats
FBLOCK	Block data, spanning input records
IEBCOPY	Process an MVS unloaded data set
PACK	Compact data
UNPACK	Unpack a previously packed stream

Printer Files

ASATOMC	Convert ASA carriage control to machine carriage control
C14TO38	Replace a set of overstruck characters with a single character
MCTOASA	Convert machine carriage control to ASA carriage control
OPTCDJ	Generate a Table Reference Character (TRC) byte
OVERSTR	Process overstruck lines
XPNDHI	Highlight spaces between words

Appendix Two: Filters

Record Format

APLDECod	Translate characters as done when read from a 3270 display
APLENCod	Translate characters as done when written to a 3270 display
CHANGE	Substitute one string for another
ESCAPE	Insert escape characters in records
REVERSE	Reverse the contents of records on a character-by-character
SCM	Line up comments and completes unclosed comments in REXX or C programs
SPECS	Rearrange the contents of records
SPILL	Split long lines at word boundaries
SNAKE	Build a multicolumn page layout
UNTAB	Expand tab characters (X'05') to blanks for lining up data into columns
XLATE	Translate characters based on a specified translation table

Appendix Two: Filters

Selection Filters

• ALL	• Select records containing a specified string or specified strings defined by an expression comprising of character strings and logical operators
• ASMFIND	• Select Assembler statements that begin with a specified text
• ASMNFIND	• Select Assembler statements that do not begin with a specified text
• BETWEEN	• Selects records between two specified targets including the records containing the target
• CASEI	• Selects records relative to a target character string regardless of case
• DROP	• Discards one or more records
• FIND	• Select records that begin with a specified text
• FRLABEL	• Select records that follow a specified target including the target record
• FRTARGET	• Select records starting with the first on selected by a specified stage
• GETRANGE	• Use in REXX program to extract part of a record to be processed
• INSIDE	• Select records between two specified targets NOT including the records containing the target

Appendix Two: Filters

Selection Filters (continued)

• LOCATE	• Select records that contain a specified string of characters
• NFIND	• Select records that do not begin with a specified text
• NINSIDE	• Select records not located between two specified targets. The records containing the targets are also selected.
• NLOCATE	• Select records that do *not* contain a specified string
• OUTSIDE	• Select records NOT located between two specified targets. The records containing the targets are not selected.
• PICK	• Compare a field in the primary input stream to a specified string or a second field in the record
• PREDSELECT	• Copies a record from its primary input stream to either its primary or secondary output stream depending on the order of arrival of input records on its other input streams.
• SCANRANGE	• Establish a token to be used by GETRANGE to parse and argument string
• SCANSTRING	• Parse an argument string containing a delimitedString
• TAKE	• Select one or more records from the beginning or end of the primary input stream
• TOLABEL	• Select records that precede a specified target, not including the target record
• TOTARGET	• Select records up to but not including the first record selected by a specified stage

Appendix Two: Filters

Selection Filters (continued)

• UNIQUE	• Compare the contents of adjacent records and discards or retains the duplicate records
• WHILELABEL	• Select consecutive records that begin with a specified string. The records must be at the beginning of the input stream.
• STRASMFIND	• Select Assembler statements that begin with a specified string of characters
• STRASMNFIND	• Select Assembler statements that do NOT begin with a specified string of characters
• STRFIND	• Select records that begin with a specified string of characters
• STRFRLABEL	• Select records that follow a specified target including the target record
• STRNFIND	• Select records that do not begin with a specified string of characters
• STRTOLABEL	• Select records that precede a specified target, not including the target record
• STRWHILELABEL	• Select consecutive records that begin with a specified string. The records must be at the beginning of the input stream.
• ZONE	• Define a range of columns from which records are selected

Appendix Two: Filters

Other Filters

• COUNT	• Count bytes, blank-delimited character strings, records, or the length of the longest or shortest line
• CRC	• Compute a checksum on the input stream
• DATECONVert	• Perform timestamp conversion and validation
• DUPLICATE	• Copy each input record a specified number of times
• FMTFST	• Formats a file status table (FST) entry
• TOKENIZE	• Parse records according to a specified list of tokens
• VCHAR	• Recode characters to a different number of bits per character

Appendix Three: Other Stages

Control Stages

ADDPIPE	Add one or more Pipelines to a set of running Pipelines
ADDSTREAm	Define unconnected input stream, output stream or both for a stage
APPEND	Write primary input stream records to the primary output stream followed by records from a specified device driver
CALLPIPE	Run a Pipeline and wait until it completes
COMMIT	Increase a stage's commit level
CONFIGURE	Change certain characteristics of Pipelines
GATE	End portions of a pipeline
MAXSTReam	Determine the highest numbered input or output stream
NOCOMMIT	Prevent automatic commit to level 0 on READTO, PEEKTO, etc
PIPCMD	Issue primary input stream records as pipeline subcommands
PIPESTOP	Terminate stages waiting for an external event
PREFACE	Write records from a specified device driver to the primary output stream followed by primary input stream records
RESOLVE	Determine if a stage is contained within and attached filter package

Appendix Three: Other Stages

Control Stages

- REXX | REXXCMD
- RUNPIPE
- SELECT
- SEVER
- SHORT
- STAGENUM
- STREAM
- STREAMSTate

- Invoke a REXX program as a stage command
- Issue input stream records as pipelines
- Select input and/or output streams to be used on subsequent READTO, PEEKTO, SEVER, OUTPUT
- Disconnect from the currently selected stream
- Connect the currently selected input stream to the currently connected output stream
- Get the stage's relative position in the pipeline
- Specify a stream by name and have the stream number returned
- Determine the state of the specified stream

Appendix Three: Other Stages

Gateways (pipe fittings)

• COLLATE	• Match records from two input streams and writes matched and unmatched records to different output streams
• DEAL	• Read records from primary input stream and write records to all connected output streams either sequentially or based on secondary input stream
• FANIN	• Combine multiple input streams into a single stream in a specified order
• FANINANY	• Combine multiple input streams into a single stream. FANINANY reads an input record from any input stream that has a record available.
• FANOUT	• Copy primary input stream records to multiple output streams
• GATHER	• Read records from all input streams and write to primary output
• JUXTAPOSE	• Prefix records in the secondary input stream with records from the primary input stream
• LOOKUP	• Find records in a reference

Appendix Three: Other Stages

Gateways (continued)

- MERGE
- NOT
- OVERLAY
- SYNCHRONIZE
- UPDATE

- Combine records from up to 10 input streams in ascending or descending order
- Reverse primary and secondary output streams of a specified stage command
- Read a record from each input stream and merges the records read into a single record
- Read records from each of its input streams while each stream has a record available
- Modify the primary input stream based on the contents of the secondary input stream

Appendix Three: Other Stages

Host Command Processors

• CMS	• Issue CMS commands with full command resolution
• COMMAND	• Issue CMS commands as if they were invoked using **Address 'COMMAND'** from REXX/VM
• CP	• Issue CP commands
• LDRTBLS	• Run a compiled REXX or Assembler user-written stage that was loaded with CMS LOAD
• NUCEXT	• Call a nucleus extension as a stage
• STARMONITOR	• Write monitor records from the CP *MONITOR system service
• STARMSG	• Write lines from a CP message service
• STARSYS	• Write lines from and sends replies to a CP system service
• SUBCOM	• Pass specified commands to a specified subcommand environment
• UDP	• Allow access to a TCP/IP port

Appendix Three: Other Stages

TCP/IP

• HOSTBYADdr	• Return DNS names for IP addresses in stream
• HOSTBYNAme	• Return IP addresses for host names in stream
• HOSTID	• Return IP address for TCP/IP stack
• HOSTNAME	• Return host name for TCP/IP stack
• IP2SOCKA	• Convert human-readable port number and IP address to a special sixteen-byte hexadecimal record for UDP
• SOCKA2IP	• Convert sixteen-byte hexadecimal record to human-readable port number and IP address
• TCPCLIENT	• Connect to a TCP/IP server
• TCPDATA	• Read from and write to a TCP/IP socket
• TCPLISTEN	• Listen on a TCP port

Appendix Three: Other Stages

Assembler

- PIPCMDM
- PIPCOMMT
- PIPDESC
- PIPEPVR
- PIPGFMOD
- PIPGFTXT
- PIPINPUT
- PIPLOCAT
- PIPOUTP
- PIPSEL
- PIPSEVER
- PIPSHORT
- PIPSTRNO
- PIPSTRST

- These are a group of Assembler macros designed to make it easier to write Pipeline extensions in Assembler. I will leave researching the specifics up to you.

  There is a good article on the Pipelines web site at: http://pucc.princeton.edu/~pipeline/sh842610.3820pack

Appendix Three: Other Stages

Service Programs

?	Display a message that describes how to obtain CMS HELP information for CMS Pipelines
AHELP	View the author's online help
FULLSCRQ	Query 3270 device characteristics
FULLSCRS	Format output from FULLSCRQ
HELP	Obtain online information about CMS Pipelines stage commands, pipeline subcommands, CMS messages, and SQL/DS return codes and commands
QUERY	Display one of the following • current version of CMS Pipelines • message level • list of messages that have been issued • current level of CMS Pipelines
TIMESTAMP	Determine when a record was read

DUMPVARS EXEC

```
/*%I:Dump variable(s) to the screen (sub routine)                :I%*/
Address 'COMMAND'
Parse Arg variable_names
Do var_index = 1 to Words(variable_names)
  variable_name = Word(variable_names,var_index)
  'PIPE VAR' variable_name '1 | VAR' variable_name
  Say variable_name '=' Value(variable_name)
  End
Return


/* test DUMPVARS*/
Address 'COMMAND'
x='Hi!'
y='Y''all'
Call DumpVars('X Y')


x = Hi!
y = Y'all
```

DUMPREXX EXEC

```
/* */
Address 'COMMAND'
Arg opt .
'PIPE CP QUERY SET |',
  'TAKE 1 |',
  'CHANGE /,/ / / |',
  'SPECS W 6 1 |',
  'VAR EMSGSET'
'CP SET EMSG OFF'
'ERASE DUMPREXX LISTING A'
rc = 0
Do i = 0 Until rc \= 0
  'PIPE (end ? name DUMPREXX)|',
     'REXXVARS' i '|',
     'VARS: TAKE 1 |',
     'SPECS "Command:" 1 W 2-* NW |',
     'APP: FANIN |',
     '>> DUMPREXX LISTING A',
   '? VARS: |',
     'BUFFER |',
     'CHANGE 1.2 /n / // |',
     'CHANGE 1.2 /v / /=/ |',
     'JOIN 1 |',
     'APP:'
  End
'CP SET EMSG' emsgset
If opt = '' Then 'XEDIT DUMPREXX LISTING A'
```

```
Command: CMS SUBROUTINE DUMPREXX EXEC L1 DUMPREXX CMS
I=0
RC=0
EMSGSET=ON
OPT=
Command: CMS COMMAND REXX EXEC D1 rexx CMS
Y=Y'all
COMMAND=x='Hi!'; y='Y''all';Call DumpREXX
X=Hi!
Command: CMS COMMAND EXECUTE XEDIT * EXECUTE XEDIT
$A=A
$D=D
$G=G
$L=L
$M=M
$N=N
$P=P
$T=T
$V=V
$X=X
ASTER=*
ASTRING=
ASTRING2=
C=
CMD=x DUMPREXX EXEC L1
CMD_CHAR1=X
...
```