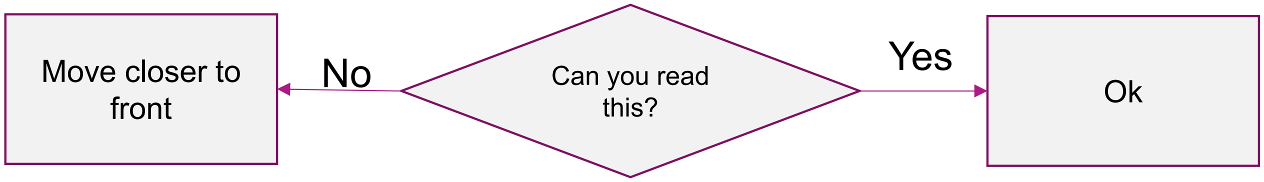
Watch and Learn: z/VM CMS Pipelines



Bill Bitner
bitnerb@us.ibm.com

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml: AS/400, DBE, e-business logo, ESCO, eServer, FICON, IBM, IBM Logo, iSeries, MVS, OS/390, pSeries, RS/6000, S/30, VM/ESA, VSE/ESA, Websphere, xSeries, z/OS, zSeries, z/VM

The following are trademarks or registered trademarks of other companies

Lotus, Notes, and Domino are trademarks or registered trademarks of Lotus Development Corporation
Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries
LINUX is a registered trademark of Linus Torvalds
UNIX is a registered trademark of The Open Group in the United States and other countries.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.
Intel is a registered trademark of Intel Corporation
* All other products may be trademarks or registered trademarks of their respective companies.

NOTES:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use.

The information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Permission is hereby granted to SHARE to publish an exact copy of this paper in the SHARE proceedings. IBM retains the title to the copyright in this paper, as well as the copyright in all underlying works. IBM retains the right to make derivative works and to republish and distribute this paper to whomever it chooses in any way it chooses.

Notice Regarding Specialty Engines (e.g., zIIPs, zAAPs and IFLs):

Any information contained in this document regarding Specialty Engines ("SEs") and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”).

No other workload processing is authorized for execution on an SE.

IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

The Basics

- CMS Pipelines is a programming framework that is very powerful. Like its name, and unix pipes, it allows data to flow through different pipes, connectors, filters, etc. We typically call each section of Pipelines a “Stage”.
  - Program
  - Arguments
  - Stage Separator

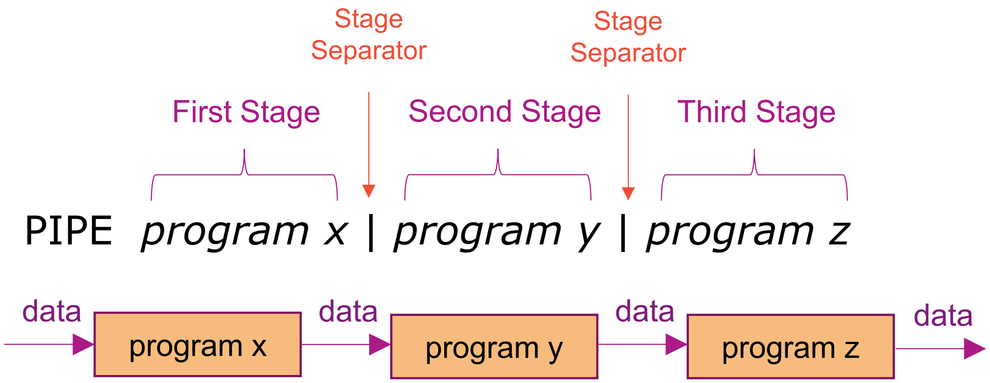
- Sources of data:
  - Constants
  - Files
  - Xedit
  - Other pipes
  - z/VM system services

- Successful programmers use “Pipe Think”
  - Breaking down the problem into various steps in manipulating the data

- Most common separator or connector is the vertical bar “|”

CMS Pipelines Implementation and Terminology

- CMS command called PIPE
  - Pipeline scanner analyses the pipeline specification
    - Multiple “stages” separated by a “stage separator” (the pipe character)
    - Each stage specifies the program and its arguments
  - Pipeline dispatcher invokes programs as specified by the pipeline
    - Not ordinary CMS programs, but specifically designed for use in pipes
    - CMS Pipelines built-in programs and user-written programs
    - Runs the programs while pumping the data through the pipeline



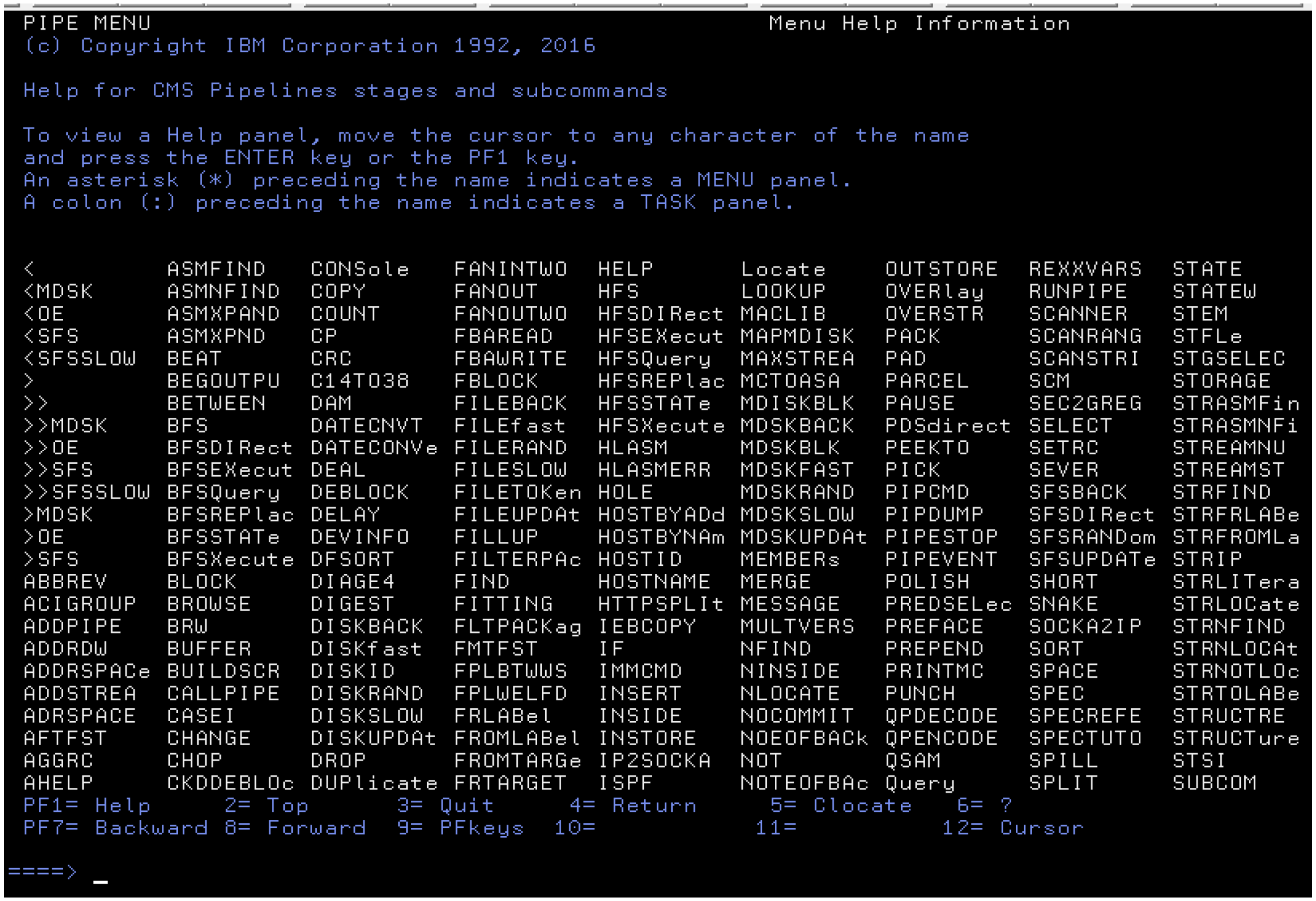
Everyone’s First Program

- Simple example:
  - – Pipe Literal Hello World | Cons
    - “Literal”: means what ever follows is data for pipe
    - “Cons”: direct data to the console
- Some output doesn’t delay, e.g.
  - – Pipe Literal Hello World | cons | > Hello World A

```
pipe literal Hello World | cons
Hello World
Ready;
```

Help Me!

- Help Pipe
  - Basic control on the Pipelines command
- Help Pipe Menu
  - Gives menu listing the various stage programs
- Help pipe *stage*
  - Help on individual stage
  - Includes examples!
- Reference Book includes help plus lists related stages



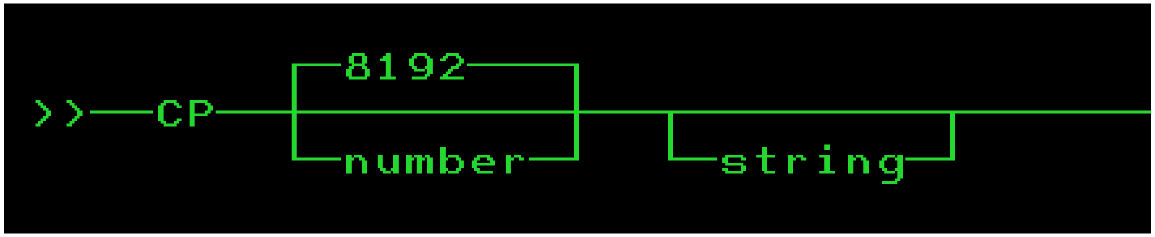
```
PIPE MENU                                           Menu Help Information
(c) Copyright IBM Corporation 1992, 2016

Help for CMS Pipelines stages and subcommands

To view a Help panel, move the cursor to any character of the name
and press the ENTER key or the PF1 key.
An asterisk (*) preceding the name indicates a MENU panel.
A colon (:) preceding the name indicates a TASK panel.


<         ASMFIND   CONSole   FANINTWO  HELP      Locate    OUTSTORE  REXXVARS  STATE
<MDSK     ASMNFIND  COPY      FANOUT    HFS       LOOKUP    OVERLay   RUNPIPE   STATEW
<OE       ASMXPAND  COUNT     FANOUTWO  HFSDIRect MACLIB    OVERSTR   SCANNER   STEM
<SFS      ASMXPND   CP        FBAREAD   HFSEXecut MAPMDISK  PACK      SCANRANG  STFLe
<SFSSLOW  BEAT      CRC       FBAWRITE  HFSQuery  MAXSTREA  PAD       SCANSTRI  STGSELEC
>         BEGOUTPU  C14TO38   FBLOCK    HFSREPlac MCTOASA   PARCEL    SCM       STORAGE
>>        BETWEEN   DAM       FILEBACK  HFSSTATe  MDISKBLK  PAUSE     SEC2GREG  STRASMFin
>>MDSK    BFS       DATECNVT  FILEfast  HFSXecute MDSKBACK  PDSdirect SELECT    STRASMNFi
>>OE      BFSDIRect DATECONVe FILERAND  HLASM     MDSKBLK   PEEKTO    SETRC     STREAMNU
>>SFS     BFSEXecut DEAL      FILESLOW  HLASMERR  MDSKFAST  PICK      SEVER     STREAMST
>>SFSSLOW BFSQuery  DEBLOCK   FILETOKen HOLE      MDSKRAND  PIPCMD    SFSBACK   STRFIND
>MDSK     BFSREPlac DELAY     FILEUPDAt HOSTBYADd MDSKSLOW  PIPDUMP   SFSDIRect STRFRLABe
>OE       BFSSTATe  DEVINFO   FILLUP    HOSTBYNAm MDSKUPDAt PIPESTOP  SFSRANDom STRFROMLa
>SFS      BFSXecute DFSORT    FILTERPAc HOSTID    MEMBERs   PIPEVENT  SFSUPDATe STRIP
ABBREV    BLOCK     DIAGE4    FIND      HOSTNAME  MERGE     POLISH    SHORT     STRLITera
ACIGROUP  BROWSE    DIGEST    FITTING   HTTPSPLIt MESSAGE   PREDSELec SNAKE     STRLOCate
ADDPIPE   BRW       DISKBACK  FLTPACKag IEBCOPY   MULTVERS  PREFACE   SOCKA2IP  STRNFIND
ADDRDW    BUFFER    DISKfast  FMTFST    IF        NFIND     PREPEND   SORT      STRNLOCAt
ADDRSPACe BUILDSCR  DISKID    FPLBTWWS  IMMCMD    NINSIDE   PRINTMC   SPACE     STRNOTLOc
ADDSTREA  CALLPIPE  DISKRAND  FPLWELFD  INSERT    NLOCATE   PUNCH     SPEC      STRTOLABe
ADRSPACE  CASEI     DISKSLOW  FRLABel   INSIDE    NOCOMMIT  QPDECODE  SPECREFE  STRUCTRE
AFTFST    CHANGE    DISKUPDAt FROMLABel INSTORE   NOEOFBACk QPENCODE  SPECTUTO  STRUCTure
AGGRC     CHOP      DROP      FROMTARGe IP2SOCKA  NOT       QSAM      SPILL     STSI
AHELP     CKDDEBLOc DUPlicate FRTARGET  ISPF      NOTEOFBAc Query     SPLIT     SUBCOM
PF1= Help     2= Top       3= Quit      4= Return      5= Clocate   6= ?
PF7= Backward 8= Forward   9= PFkeys   10=            11=          12= Cursor

====> _
```

CP Command Output in Pipelines

- Use the “CP” stage followed by a command

```
pipe cp query time | cons
TIME IS 10:48:19 EDT MONDAY 10/09/17
CONNECT= 99:59:59 VIRTCPU= 000:01.61 TOTCPU= 000:02.47
Ready;
```

- CP Syntax
  - Number deals with how big a buffer to create for commands with very large output



```
          ┌─8192───┐
>>───CP───┼────────┼───┬────────┬────
          └─number─┘   └─string─┘
```

Problem 1: I Want to know how many Page Volumes I Have?

- I could use QUERY ALLOC PAGE and count, but I don’t like to count

```
                 EXTENT      EXTENT  TOTAL  PAGES   HIGH    %
VOLID  RDEV       START         END  PAGES IN USE   PAGE USED
------ ---- ----------- ----------- ------ ------ ------ ----
PGG700 2700           1       10016  1761K  76764 165699   4%
PGG701 2701           1       10016  1761K  82926 179154   4%
PGG702 2702           1       10016  1761K  79160 174218   4%
PGG703 2703           1       10016  1761K  83376 181899   4%
PGG704 2704           1       10016  1761K  82471 174240   4%
PGG705 2705           1       10016  1761K  80245 168644   4%
PGG706 2706           1       10016  1761K  87295 186236   4%
PGG707 2707           1       10016  1761K  77462 177829   4%
PGG708 2708           1       10016  1761K  83676 187200   4%
…
PGG734 2722           1       10016  1761K  84817 176826   4%
                                    ------ ------        ----
SUMMARY                             61622K  2817K          4%
USABLE                              61622K  2817K          4%
Ready;
```

Problem 1: I Want to know how many Page Volumes I Have?

```
PIPE CP q alloc page | > alloc page a
Ready;
```

- I could then look in the file ALLOC PAGE A and count, but I don’t like to count that either.

```
PIPE CP q alloc page | COUNT lines | CONS
41
Ready;
```

- Wait, that’s wrong, I forgot about the headers and summary lines, we have to remove 3 from top and 3 from bottom.

```
PIPE CP q alloc page | DROP 3 | DROP LAST 3 | COUNT LINES | CONS
35
Ready;
```

Problem 2: Which real volumes contain my virtual disks

- I could use QUERY DASD and cut and paste, but I am lazy

```
QUERY V DASD
DASD 009B 3390 USE724 R/O         10 CYL ON DASD  D666 SUBCHANNEL = 0017
DASD 0120 3390 SYE711 R/O        250 CYL ON DASD  D548 SUBCHANNEL = 0016
DASD 0190 3390 USG7CB R/O        214 CYL ON DASD  2524 SUBCHANNEL = 000F
DASD 0191 3390 USG72A R/W        300 CYL ON DASD  D57D SUBCHANNEL = 0008
DASD 019B 3390 USE740 R/O        300 CYL ON DASD  E360 SUBCHANNEL = 0012
DASD 019D 3390 US7E53 R/O        250 CYL ON DASD  E375 SUBCHANNEL = 0010
DASD 019E 3390 USG7AO R/O        400 CYL ON DASD  DB3F SUBCHANNEL = 0011
DASD 019F 3390 USE73L R/O        100 CYL ON DASD  D76E SUBCHANNEL = 0013
DASD 01A1 3390 US7EA6 R/O        100 CYL ON DASD  C703 SUBCHANNEL = 0015
DASD 0223 3390 USP749 R/W         22 CYL ON DASD  D60F SUBCHANNEL = 0000
…
DASD 02BD 3390 USP773 R/W       2000 CYL ON DASD  D50B SUBCHANNEL = 0005
DASD 0399 3390 USP749 R/O         30 CYL ON DASD  D60F SUBCHANNEL = 0014
DASD 0419 3390 USE71F R/W         17 CYL ON DASD  D748 SUBCHANNEL = 0003
DASD 0A91 3390 USE719 R/O         10 CYL ON DASD  D745 SUBCHANNEL = 0018
Ready;
```

Problem 2: Which real volumes contain my virtual disks

- Let’s use Pipelines and a new very powerful stage called “SPEC”
- SPEC has many options, one is to parse different words
  - Here we take the 10th word and place it in column 1
  - The real device address was the 10th word

```
PIPE CP QUERY V DASD | SPEC w10 1 | cons
D666
D548
2524
D57D
E360
E375
DB3F
D76E
C703
D60F
…
D50B
D60F
D748
D745
Ready;
```

Problem 2: Which real volumes contain my virtual disks

- How many have duplicate Volumes? Lets use SPEC and COUNT to determine total number of virtual DASD

```
PIPE CP QUERY V DASD | SPEC w10 | COUNT LINES | CONS
21
Ready;
```

- Now use a new SORT option to only get the UNIQUE ones

```
PIPE CP QUERY V DASD | SPEC w10 | SORT UNIQUE  | COUNT LINES | CONS
20
```

- So there is one real device that has two virtual DASD on it (21 – 20 = 1)

Problem 2: Which real volumes contain my virtual disks

- What if we want more details? Use SORT COUNT.

```
PIPE CP QUERY DA | SPEC w10 2 | SORT COUNT | CONS
        1 C703
        1 DA15
        1 DB3F
        1 D44A
        1 D50B
        1 D548
        1 D57D
        2 D60F
        1 D617
        1 D664
        1 D666
        …
        1 E30F
        1 E360
        1 E370
        1 2524
Ready;
```

- One Volume (D60F) has two vdevs on it, rest have one
- Notice:
  - Moved w10 from column 1 in previous example to column 2 in order to make more readable here
  - SORT COUNT – sorts the input records and in the process removes duplicates. The COUNT option of SORT keeps count of total records matching this.

Problem 2: Which real volumes contain my virtual disks

- But I don’t want this on my console, especially for large machines so I will put in a CMS file

```
PIPE CP QUERY V DASD | SPEC w10 2 | SORT COUNT | > virtreal dasdlist a
Ready;
```

- The CONS stage is replaced the “>” which indicates output goes to file id that follows, in this case “virtreal dasdlist a” file
- You can use both CONS and a “>” director. I do this sometimes for validation

```
PIPE CP QUERY V DASD | SPEC w10 2 | SORT COUNT | > virtreal dasdlist a | CONS
```

- “>” creates a new file of that name even if one exists
- “>>” appends to file of that name if it exists, otherwise creates new one
- “<“ allows you to read from a file on other end of pipe

Use as a REXX Exec

```
/* BITQVIR EXEC                                         */
/* Bit's Virtual on Real Dasd List Exec                 */
'PIPE CP QUERY V DASD',    /* Get list of virtual DASD */
'| SPEC W10 2',            /* Real address is word 10  */
'| SORT COUNT',            /* Find duplicates          */
'| SORT 1-10 DESCEND',     /* Sort descendig on count  */
'| > virtreal dasdlist a' /* Write out results        */
exit
```

- Use continuation character, the comma.
- Start with a connector
- Use comments
- Introduced new SORT stage with DESCEND option to sort in descending order based on columns 1-10

Two Nice features for REXX – First VAR

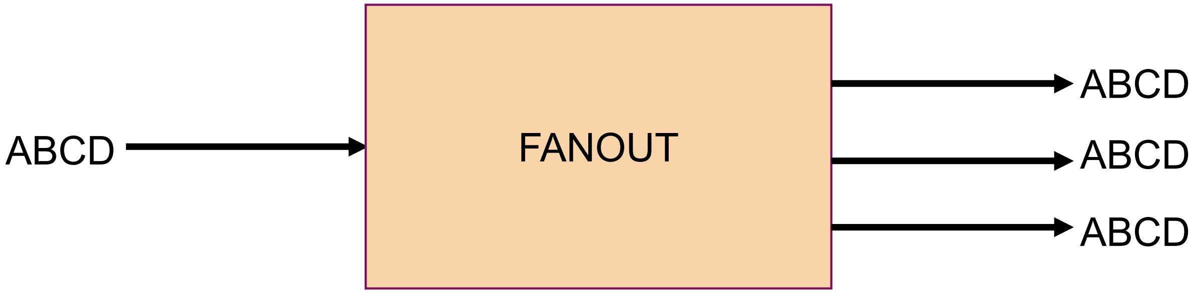
```
/* BITQDVAR - Example put number of Devices in a variable       */
'PIPE CP QUERY V DASD',                /* Get list of virtual DASD */
'| SPEC W10 2',                        /* Real address is word 10  */
'| SORT COUNT',                        /* Find duplicates          */
'| COUNT LINES',                       /* Count of real devices    */
'| VAR num_real_devices’               /* store in variable        */
Say "Number Real:" num_real_devices /* now can use as variable  */


exit
```

```
BITQDVAR
Number Real: 20
Ready;
```

Two Nice features for REXX – Second STEM

```
/* BITQDSTM - REXX Stem variable example with Real device addresses  */
'PIPE CP QUERY V DASD',               /* Get list of virtual DASD    */
'| SPEC W10 1',                       /* Real address is word 10     */
'| STEM' real_device.                 /* Put the lines into a Stem   */

Do i = 1 to real_device.0             /* .0 is number of entries     */
   Say 'A device is:' real_device.i   /* Do something with it        */
End /* For each real device */

exit
```

Multistream Pipes – Example Fanout



As in real life, plumbing often involves more than a single straight path.

Multi-streams – A few Things to Know

- Most often done inside an Exec.
- Need a way to mark the end of streams
  - PIPE (endchar ?)
- Need a way to mark where streams connect
  - Labels, for simple pipes a character followed by colon (e.g. “f:”)
- Examples in the Help are your friend!

Problem Three: Determine Entitlement of a Logical Partition

- Without using performance data.
- Multiple ways to solve this
- Leverage, the CP command QUERY PROC TOPOLOGY

```
Q PROC TOPOLOGY
TOPOLOGY
  NESTING LEVEL: 02  ID: 01
    NESTING LEVEL: 01  ID: 01
      PROCESSOR 00  MASTER     CP    VH  0000
      PROCESSOR 01  ALTERNATE  CP    VH  0001
      PROCESSOR 02  ALTERNATE  CP    VM  0002
      PROCESSOR 03  ALTERNATE  CP    VL  0003
Ready;
```

- 2 Vertical High, 1 Vertical Medium, 1 Vertical Low
  - High = 100, Low = 0, Medium = something else

Problem Three: Determine Entitlement of a Logical Partition

- Logic
  - Find out how many vertical highs, how many vertical mediums, and how many vertical lows
  - Do math on those counts 100 x VH + 75 x VM + 0 x VL

- I can use a similar approach but I need to do it for all three (well really just VH and VM)

- Use LOCATE stage to find a record that contains a string

- Use VAR to save the counts and do the math

Problem Three: Determine Entitlement of a Logical Partition

```
/* Determine rough entitlement */
'PIPE (end ?)',
'| CP Q PROC TOPOLOGY', /* Get topology Info          */
'| f: fanout',          /* fanout to all the streams  */
'| locate /VH/',        /* locate vertical high       */
'| count lines',        /* count lines with them      */
'| VAR VH',             /* store count in variable VH */
'?f:',                  /* second stream              */
'| locate /VM/',        /* locate vertical medium     */
'| count lines',        /* count lines with them      */
'| VAR VM',             /* store count in variable VM */
'?f:',                  /* third stream               */
'| locate /VL/',        /* locate vertical low        */
'| count lines',        /* count lines with them      */
'| VAR VL'              /* store count in variable VL */
Entitlement = 0*VL + 0.75*VM + VH
Say 'Estimated entitlement is' Entitlement
```

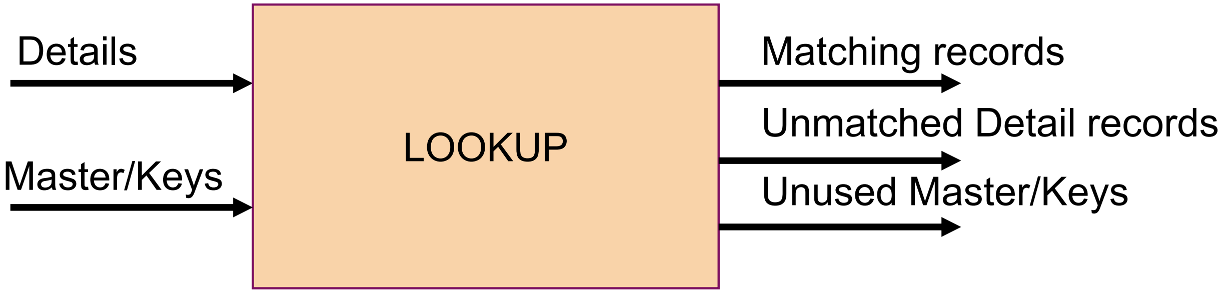
Problem Three: Determine Entitlement of a Logical Partition

```
/* Determine rough entitlement */
'PIPE (end ?)',
'| CP Q PROC TOPOLOGY',
'| f: fanout',
'| locate /VH/',
'| count lines',
'| VAR VH',
'?f:',
'| locate /VM/',
'| count lines',
'| VAR VM',
```

```
'?f:',
'| locate /VL/',
'| count lines',
'| VAR VL’
Entitlement = 0*VL + 0.75*VM + VH
Say 'Estimated entitlement is'
Entitlement
```

- No continuation after VAR VL, as that is end of Pipelines

Multistream Pipes - Lookup



Details

Master/Keys

LOOKUP

Matching records

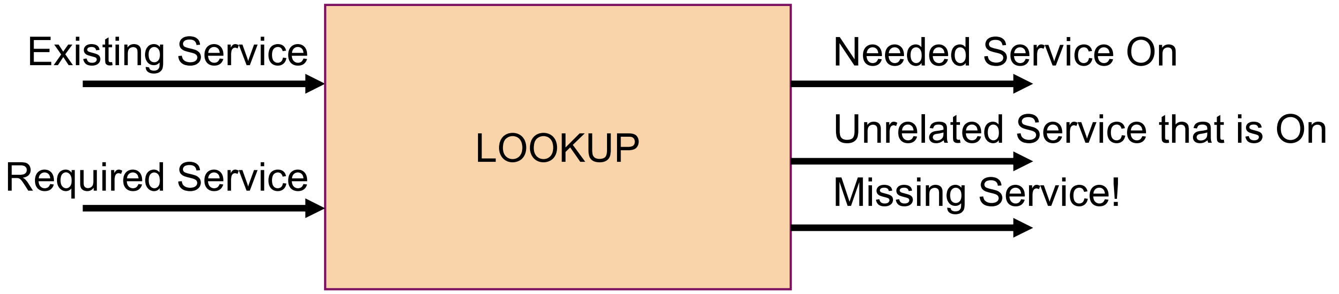
Unmatched Detail records

Unused Master/Keys

Problem Four: Do I have the Service I need?

- Given a list of z/VM APARs, such as:
  - VM65942 VM65988 VM66071 VM65867 VM65865 VM65870

- How do I tell which, if any, are missing from my z/VM system? What if the list is even larger?



Problem Four: Do I have the Service I need?

- Input streams
  - Details will come from the CP command QUERY CPSERVICE
    - Example to get all APARs applied that start with VM6 use

      CP QUERY CPSERVICE APAR VM6*
  - Our list of required APARs will come from a file.

```
===== * * * Top of File * * *

===== VM65942 VM65988 VM66071 VM65867 VM65865 VM65870

===== * * * End of File * * *
```

- Output streams
  - Matching APARs are good and on system, keep in a file.
  - Other APARs are not of interest, so throw away
    - Use HOLE stage in Pipelines for illustrative purposes
  - Missing APARs keep in a file

Problem Four: Do I have the Service I need?

```
QUERY CPSERVICE APAR VM6*
APAR        PTF
VM65355     UM34984
VM65481     UM34941
VM65644     UM34950
VM65741     UM34922
VM65752     UM34981
VM65846     UM34947
VM65860     UM34957
VM65865     UM34977
VM65866     UM34933
VM65870     UM34961
VM65871     UM34930
VM65872     UM35053
VM65877     UM34971
```

- Note the one header line
  - – Will want to DROP that before entering Lookup
- APARs in columns 1-7
  - – Will want to do lookup based on these columns

Problem Four: Getting List of Required APARs in Shape

```
===== * * * Top of File * * *
===== VM65942 VM65988 VM66071 VM65867 VM65865 VM65870
===== * * * End of File * * *
```

```
PIPE < required apars a | SPLIT | CONS
VM65942
VM65988
VM66071
VM65867
VM65865
VM65870
```

- Need one APAR per line/record
  - Use SPLIT APARs in columns 1-7
  - Will want to do lookup based on these columns

Problem Four: Do I have the Service I need?

```
/* Check for Service                                                         */
'PIPE (end ?)',
'CP QUERY CPSERVICE APAR VM6*',     /* Get all service                        */
'| Drop 1',                         /* remove header                          */
'|l: lookup 1.7',                   /* APAR number is first 7 characters      */
'| SORT UNIQUE 1-7',                /* Remove Master                          */
'|> APPLIED APARS A',               /* Applied APARs                          */
‘?< required apars a',              /* Read list of APARs                     */
'| SPLIT',                          /* Create one APAR per line               */
'|l:',                              /* Secondary streams                      */
'| HOLE',                           /* Not matched from details, just ignore  */
'?l:',                              /* Tertiary streams                       */
'|> MISSING APARS A'                /* Masters that were not referenced       */
```

Why the SORT UNIQUE 1-7?

```
===== * * * Top of File * * *
===== VM65865     UM34977
===== VM65865
===== VM65870     UM34961
===== VM65870
===== VM65942     UM35208
===== VM65942
===== * * * End of File * * *
```

- The output stream for matches includes the details and the master. We only care about the detail.
- There are other ways to accomplish this with other options on the LOOKUP stage

You can write your own stages in REXX and other Languages

- The programs have a filetype of REXX.

- Basic construction is giant loop where you pull in data from the Pipe stage in front of you and write out data in the stream from you.

- For example, we have a file that has information in inches and we want to convert to centimeters.
  - Even SPEC doesn’t do this
  - We can write a simple program

REXX Stage for Inches to Centimeters

```
/* I2C REXX: Convert Inches to Centimeters */
DO FOREVER
   "READTO nextrec"
   IF rc <> 0 THEN LEAVE
   inches = nextrec
   centimeters = inches * 2.54
   "OUTPUT" inches centimeters
END
Exit
```

```
pipe literal 1.3 10 4 2 | split | i2c |
cons
1.3 3.302
10 25.40
4 10.16
2 5.08
Ready;
```

CP System Services and Pipelines

- STARSYS stage - *ACCOUNT, *LOGREC, *SYMPTOM
- STARMON stage - *MONITOR
- STARMSG stage – Connect via IUCV to messages

```
pipe starmon mondcss shared | locate 5 x01| locate 8 x0F| spec 21.8 1 77.8 nw 93.4
c2x nw | cons

MNTDASD2 ALEXIAA  1FFFFFFF
COYLE             3FFFFFFF
HOTTENMA          03FFFFFF
MONWRITE          07FFFFFF
BRAZIE            1FFFFFFF
RIVADENE          01FFFFFF
OVVMCHEK          01FFFFFF
MEAS00            01FFFFFF
QWATCH            01FFFFFF
```

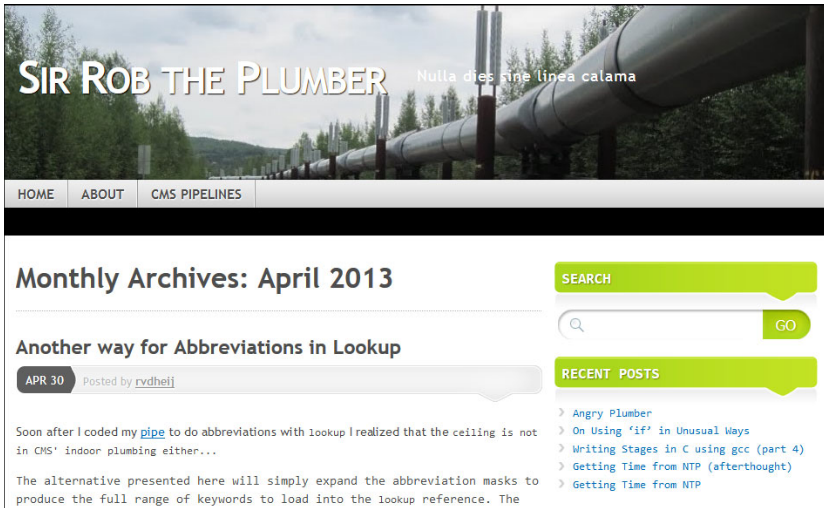
Further Reading - Introduction

- CMS Pipelines home page http://vm.marist.edu/~pipeline
  - Papers by Melinda Varian
  - CMS Pipelines Tutorial

- CMSPIP-L Mailing List
  - Subscribe through listserv@vm.marist.edu

- CMS Pipelines Author's Edition
  - Part 1. Introduction
  - Part 2. Task Oriented Guide

- z/VM CMS Pipelines User's Guide



Monthly Archives: April 2013

Another way for Abbreviations in Lookup

APR 30 Posted by rvdheij

Soon after I coded my pipe to do abbreviations with lookup I realized that the ceiling is not in CMS' indoor plumbing either...

The alternative presented here will simply expand the abbreviation masks to produce the full range of keywords to load into the lookup reference. The

RECENT POSTS

- Angry Plumber
- On Using 'if' in Unusual Ways
- Writing Stages in C using gcc (part 4)
- Getting Time from NTP (afterthought)
- Getting Time from NTP

rvdheij.wordpress.com

Summary

- Pipelines is powerful!
- Pipelines is useful!
- Pipelines is fun!