NetRexx on the Big Iron

2011 Rexx Language Symposium, Aruba

René Vincent Jansen, 2011-12-04

Agenda

- NetRexx: what is it
- NetRexx: how to get it?
- Uploading to the Mainframe
- Running the translator
    - z/OS Unix Systems Services
    - z/Linux

NetRexx: What is it?

- We do not need to explain mainframe people what Rexx is
- NetRexx is Rexx for Java
- Java runs everywhere
- For occasional scripting to full blown applications

NetRexx: How to get it

- www.netrexx.org
- Download it (one version for all platforms)
- or build it from source (it is written in NetRexx, so you’ll get the previous version to build it with)
- you’ll need a JVM

JVM on z/OS

- Try if it is there in an OMVS shell
- java -version
- Inquire at the systems programming department
  - you might need something in your profile

JVM on z/OS

- If you are the systems programmer:
  - Download from IBM
  - Do an SMP/E install or just run the installer

http://www-03.ibm.com/systems/z/os/zos/tools/java/

When you have NetRexxC.jar

- And a running JVM
- You have everything you need
- The same jar (Java archive) runs everywhere
  - This means: it is entirely portable, you just need to copy the one file you have to every platform you run it on
- Need to get it to run now on z/OS or z/Linux

Getting it on the host

- First need to get it there
- Use your favourite terminal emulator
- do an IND$FILE send - in Binary (bin) mode
- or use ftp - also binary mode
- everything you have - Connect:Direct, MQ Series
- as long as you transfer every bit of NetRexxC.jar

Place it in the HFS (zfs)

- When uploading using IND$FILE, first make a load library with lrecl 0 to contain it and copy it later
- ftp can directly put it in HFS
- place it somewhere you'll remember
- in ~/lib in your home directory
- or with other class libraries you have

The ClassPath

- Java finds Class Libraries on the class path
- it uses an environment variable CLASSPATH to establish you class path
- one catch: when compared to other paths, please know that it originally pointed to directories containing .class files
- now that we are using compressed class archives (jar files), *you need to specify the whole filename*

The binary path

- It is not strictly needed, you will just need the java executable to work
- But if you do not have the time to repeatedly type
    - java org.netrexx.process.NetRexxC [sourcefile]
- Then copy the nrc shell script to your ~/bin library and make sure it is on the $PATH

Running the translator

- The translator translates (Net)Rexx to Java
- If you use javac as the Java Compiler (and there is no reason not to, in fact it is the default) the NetRexx language processor needs to find the Java compiler class from the JDK - Java Development Kit
- It also needs to find the Java Runtime Library itself
  - (which contains the whole included class hierarchy)

The Classpath content

- This is dependent on where it is installed
- If you executed the IBM JVM 1.6 Install procedure
- Then chances are, this looks like:
- /opt/ibm/java-s390x-60/lib/tools.jar:/opt/ibm/java-s390x-60/jre/lib/s390x/default/jclSC160/vm.jar

The Good News

- if the compiler and runtime library are indeed to be found at /opt/ibm/java-s390x-60/lib/tools.jar:/opt/ibm/java-s390x-60/jre/lib/s390x/default/jclSC160/vm.jar
- then NetRexx, starting from version 3.00 (the RexxLA editions), already knows this, and you will not have to add anything

Doing the Hello World test

- To verify this installation, open a file using
  - oedit hello.nrx
  - type in: say ‘Hello, Big Iron world!’
  - save it
- run:
  - nrc hello

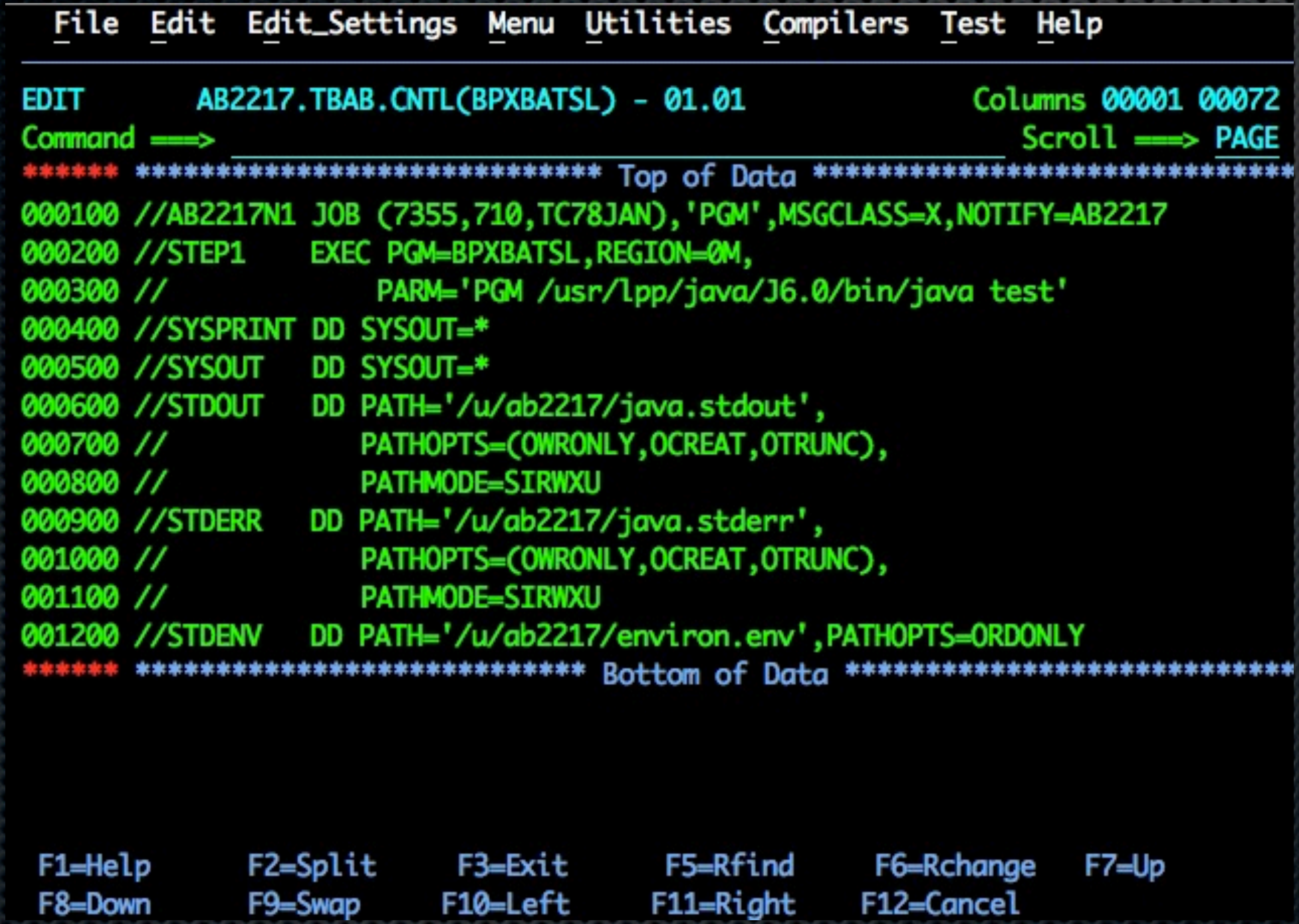
Integrating with MVS

- MVS here chosen as an easy reference to the parts of z/OS that existed before Open Edition came along - traditional z/OS
- Usage is not limited to USS and Open Edition Shell
- In fact, you can use batch programs, read MVS datasets, interact with the operators via the console, etcetera

Three ways to interact with MVS

- 1 BPXBATCH
- 2 BPXBATSL
- 3 JZOS JCL PROCEDURE

Using BPXBATSL



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       AB2217.TBAB.CNTL(BPXBATSL) - 01.01             Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100 //AB2217N1 JOB (7355,710,TC78JAN),'PGM',MSGCLASS=X,NOTIFY=AB2217
000200 //STEP1    EXEC PGM=BPXBATSL,REGION=0M,
000300 //             PARM='PGM /usr/lpp/java/J6.0/bin/java test'
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSOUT   DD SYSOUT=*
000600 //STDOUT   DD PATH='/u/ab2217/java.stdout',
000700 //            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
000800 //            PATHMODE=SIRWXU
000900 //STDERR   DD PATH='/u/ab2217/java.stderr',
001000 //            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
001100 //            PATHMODE=SIRWXU
001200 //STDENV   DD PATH='/u/ab2217/environ.env',PATHOPTS=ORDONLY
****** **************************** Bottom of Data ****************************



 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

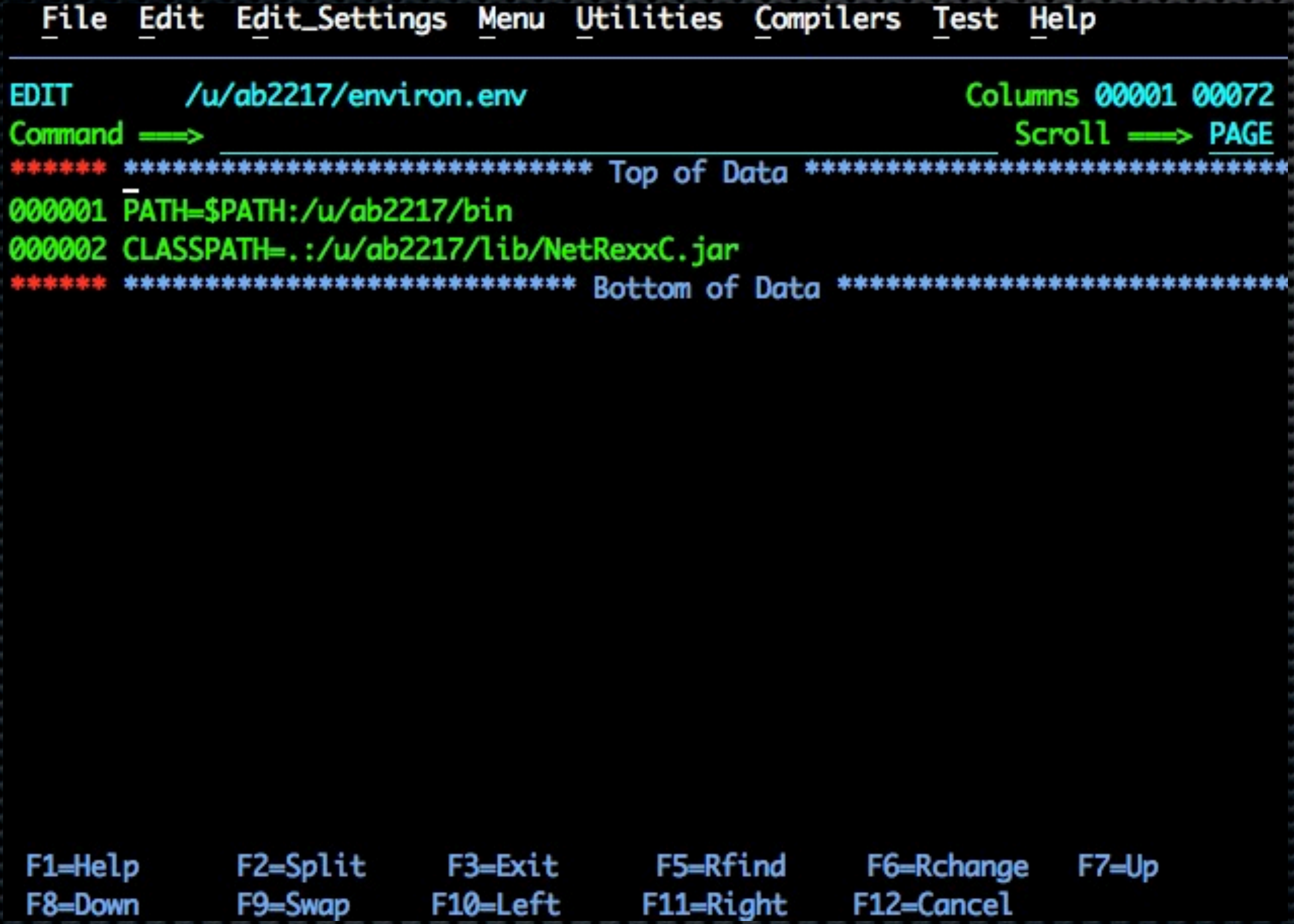
JZOS Features

- Run Java applications on z/OS seamlessly in an MVS batch job step or Started Task
- Access to datasets via JCL DD statements
- Send output to JES2 Sysout with auto-transcoding
- Pass condition codes between Java and non-Java jobsteps
- Communicate with the MVS system console

JZOS Features, Cont.

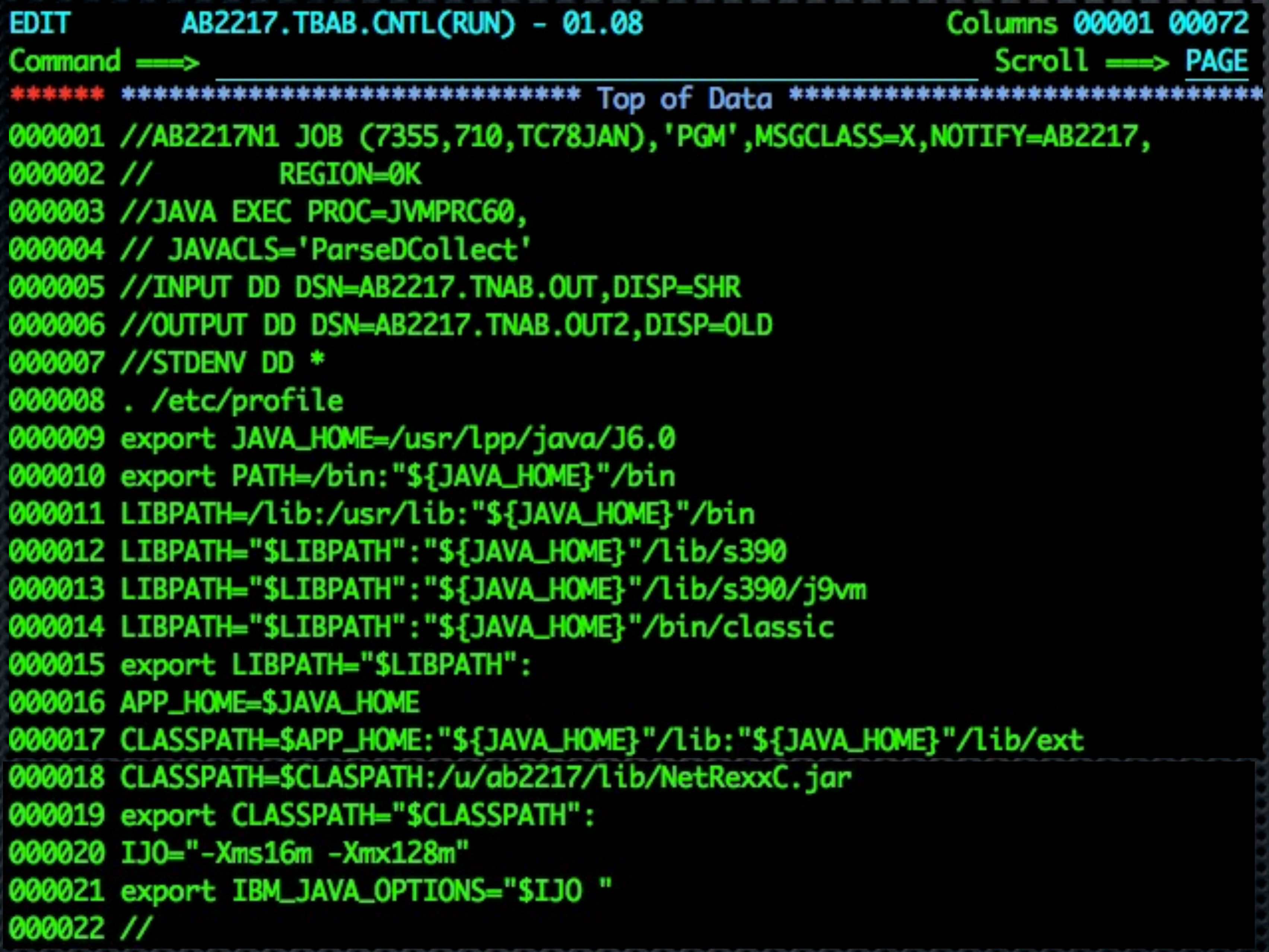
- Read and write traditional MVS datasets from Java
- Java interfaces to MVS API's like SMF, Catalog Search and Log Streams
- Classes to convert COBOL and Assembler data types to Java objects
- Invoke DFSORT, IDCAMS
- Serialize resources (ENQ) and use WLMs

Setting the environment



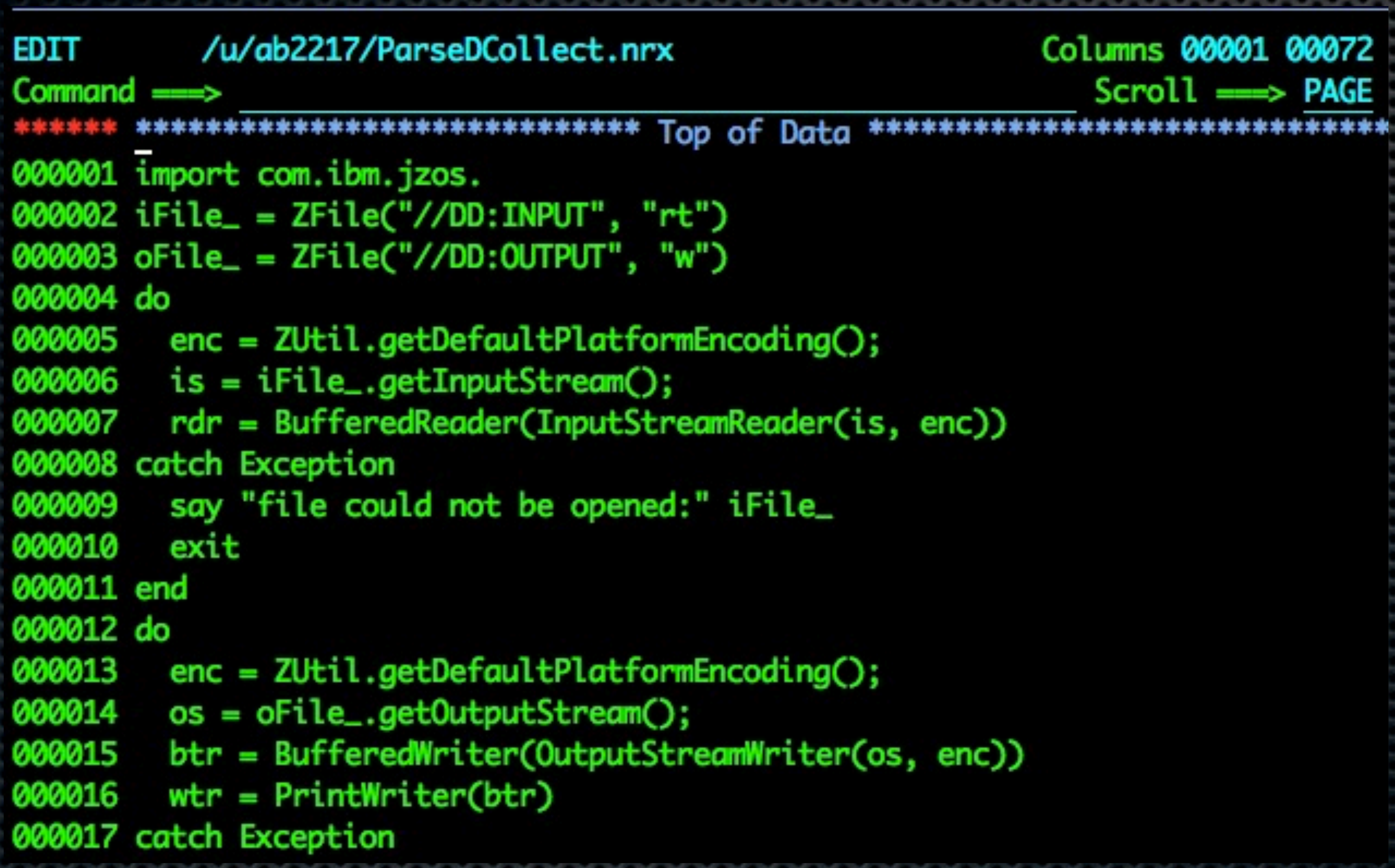
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       /u/ab2217/environ.env                          Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000001 PATH=$PATH:/u/ab2217/bin
000002 CLASSPATH=.:/u/ab2217/lib/NetRexxC.jar
****** **************************** Bottom of Data ****************************

 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Using the JZOS procedure



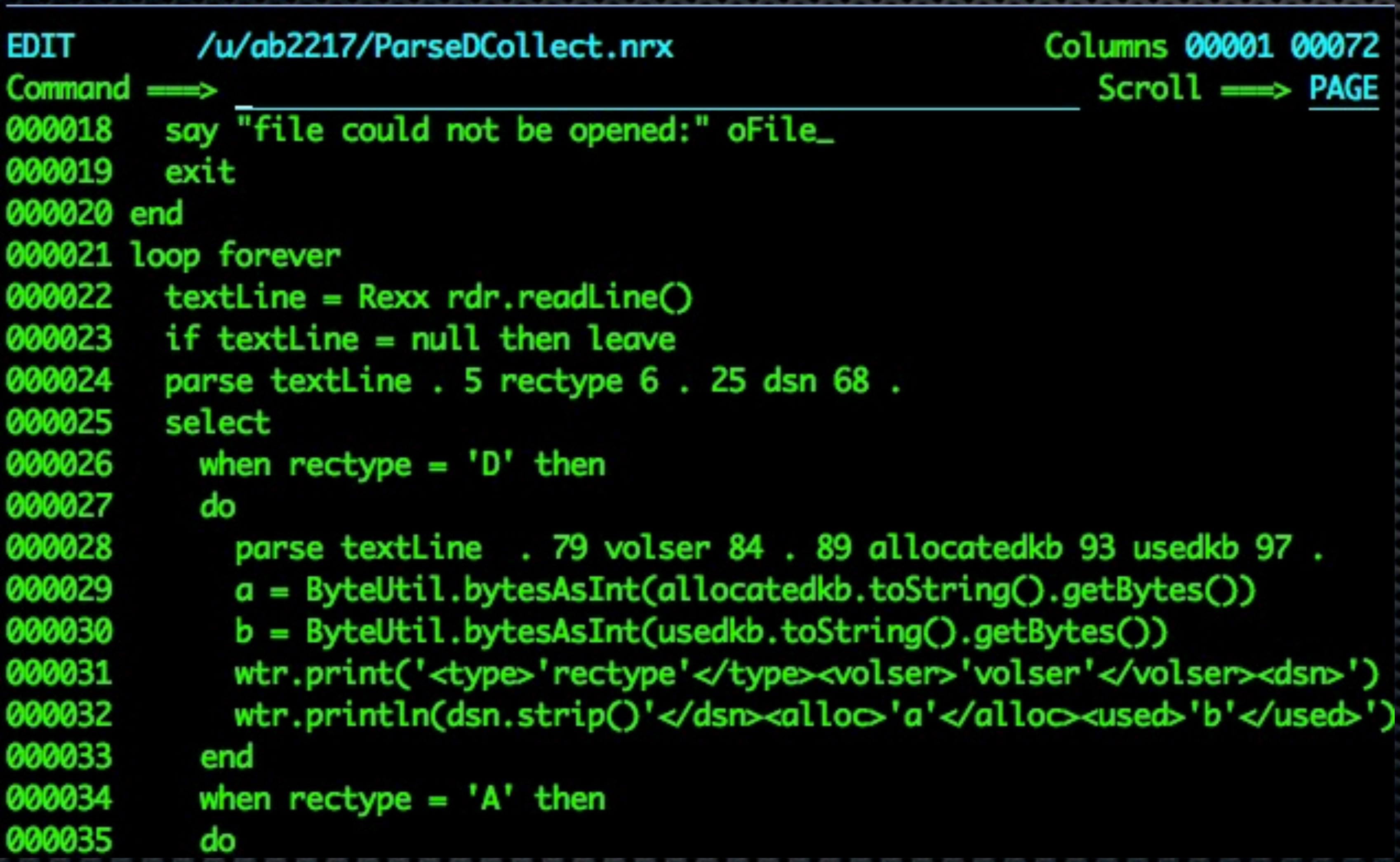
```
EDIT       AB2217.TBAB.CNTL(RUN) - 01.08                  Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000001 //AB2217N1 JOB (7355,710,TC78JAN),'PGM',MSGCLASS=X,NOTIFY=AB2217,
000002 //        REGION=0K
000003 //JAVA EXEC PROC=JVMPRC60,
000004 // JAVACLS='ParseDCollect'
000005 //INPUT DD DSN=AB2217.TNAB.OUT,DISP=SHR
000006 //OUTPUT DD DSN=AB2217.TNAB.OUT2,DISP=OLD
000007 //STDENV DD *
000008 . /etc/profile
000009 export JAVA_HOME=/usr/lpp/java/J6.0
000010 export PATH=/bin:"${JAVA_HOME}"/bin
000011 LIBPATH=/lib:/usr/lib:"${JAVA_HOME}"/bin
000012 LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390
000013 LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390/j9vm
000014 LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic
000015 export LIBPATH="$LIBPATH":
000016 APP_HOME=$JAVA_HOME
000017 CLASSPATH=$APP_HOME:"${JAVA_HOME}"/lib:"${JAVA_HOME}"/lib/ext
000018 CLASSPATH=$CLASPATH:/u/ab2217/lib/NetRexxC.jar
000019 export CLASSPATH="$CLASSPATH":
000020 IJO="-Xms16m -Xmx128m"
000021 export IBM_JAVA_OPTIONS="$IJO "
000022 //
```

Referencing MVS Datasets



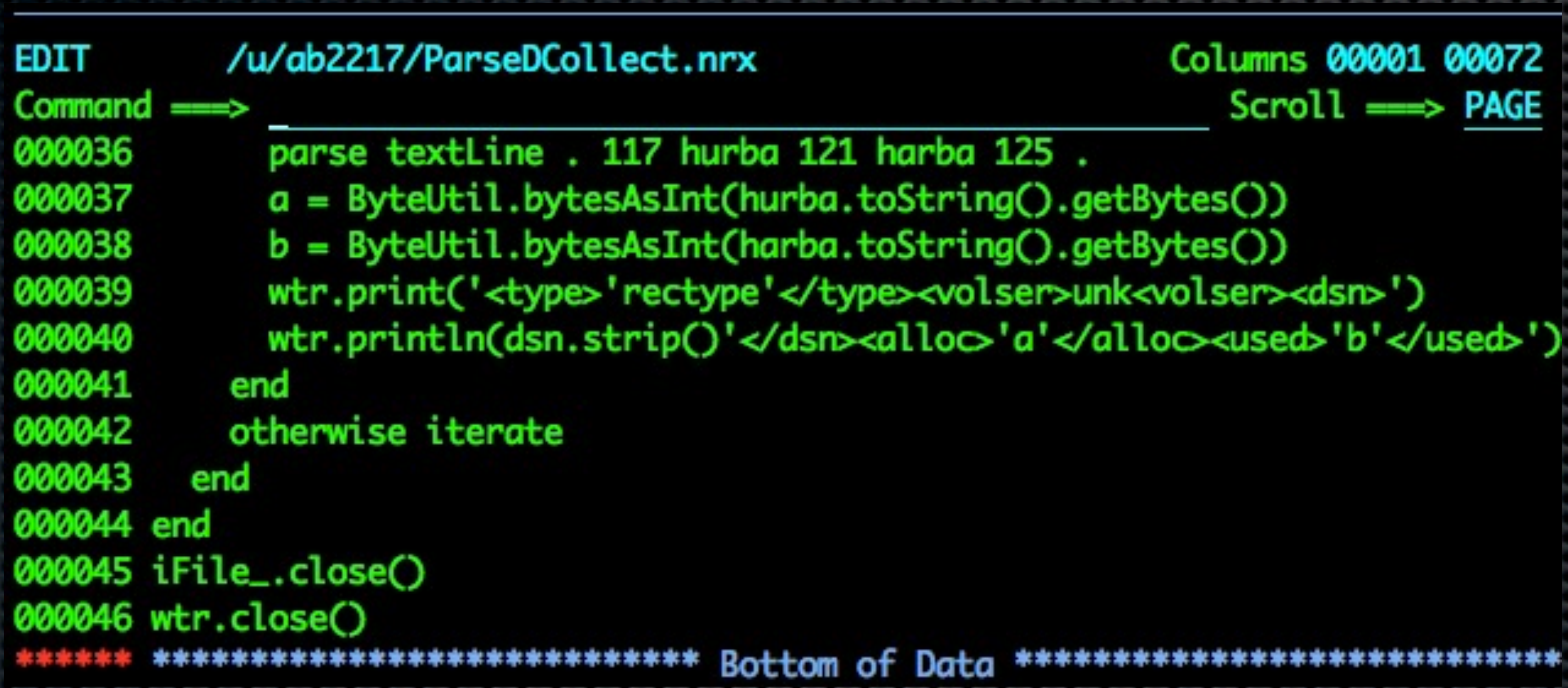
```
EDIT       /u/ab2217/ParseDCollect.nrx                   Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
import com.ibm.jzos.
iFile_ = ZFile("//DD:INPUT", "rt")
oFile_ = ZFile("//DD:OUTPUT", "w")
do
  enc = ZUtil.getDefaultPlatformEncoding();
  is = iFile_.getInputStream();
  rdr = BufferedReader(InputStreamReader(is, enc))
catch Exception
  say "file could not be opened:" iFile_
  exit
end
do
  enc = ZUtil.getDefaultPlatformEncoding();
  os = oFile_.getOutputStream();
  btr = BufferedWriter(OutputStreamWriter(os, enc))
  wtr = PrintWriter(btr)
catch Exception
```

Referencing MVS Datasets 2



```
EDIT       /u/ab2217/ParseDCollect.nrx                     Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> PAGE
  say "file could not be opened:" oFile_
  exit
end
loop forever
  textLine = Rexx rdr.readLine()
  if textLine = null then leave
  parse textLine . 5 rectype 6 . 25 dsn 68 .
  select
    when rectype = 'D' then
    do
      parse textLine  . 79 volser 84 . 89 allocatedkb 93 usedkb 97 .
      a = ByteUtil.bytesAsInt(allocatedkb.toString().getBytes())
      b = ByteUtil.bytesAsInt(usedkb.toString().getBytes())
      wtr.print('<type>'rectype'</type><volser>'volser'</volser><dsn>')
      wtr.println(dsn.strip()'</dsn><alloc>'a'</alloc><used>'b'</used>')
    end
    when rectype = 'A' then
    do
```

Referencing MVS Datasets 3



```
EDIT       /u/ab2217/ParseDCollect.nrx                   Columns 00001 00072
Command ===> _                                              Scroll ===> PAGE
      parse textLine . 117 hurba 121 harba 125 .
      a = ByteUtil.bytesAsInt(hurba.toString().getBytes())
      b = ByteUtil.bytesAsInt(harba.toString().getBytes())
      wtr.print('<type>'rectype'</type><volser>unk<volser><dsn>')
      wtr.println(dsn.strip()'</dsn><alloc>'a'</alloc><used>'b'</used>')
    end
    otherwise iterate
  end
end
iFile_.close()
wtr.close()
****** **************************** Bottom of Data ****************************
```

Discussion points

- Why do this
- Performance - zAAP assistance processors
- More for introduction of older systems staff to Java than to introduce new personnel to mainframe?

Thank you

- Questions?