Contents

The NetREXX Programming Series i

1 Introduction 1

2 The Pipeline Concept 2
2.1 What is a Pipeline? 2
2.2 Stage 2
2.3 Device Driver 3

3 Running pipelines 4
3.1 Configuration 4
3.2 From the NetREXX Workspace (nrws) with direct execution 5
3.3 From the command line with direct execution 5
3.4 Precompiled Pipelines 6
3.5 Compiled from an .njp file 6
3.6 Compiled from an .njp file with additional stage definitions in NetREXX 7

4 Example Session 8

5 Write your own Filters 12

6 More advanced Pipelines 15

7 Device Drivers 16

8 Record Selection 18

9 Filters 19

10 Other Stages 20

11 Multi-Stream Pipelines 21

12 Pipeline Stalls 23

13 How to use a pipe in a NetREXX program 25
# The NetREXX Programming Series

This book is part of a library, the *NetREXX Programming Series*, documenting the NetREXX programming language and its use and applications. This section lists the other publications in this series, and their roles. These books can be ordered in convenient hardcopy and electronic formats from the Rexx Language Association.

<table>
<thead>
<tr>
<th><strong>Quick Start Guide</strong></th>
<th>This guide is meant for an audience that has done some programming and wants to start quickly. It starts with a quick tour of the language, and a section on installing the NetREXX translator and how to run it. It also contains help for troubleshooting if anything in the installation does not work as designed, and states current limits and restrictions of the open source reference implementation.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming Guide</strong></td>
<td>The Programming Guide is the one manual that at the same time teaches programming, shows lots of examples as they occur in the real world, and explains about the internals of the translator and how to interface with it.</td>
</tr>
<tr>
<td><strong>Language Reference</strong></td>
<td>Referred to as the NRL, this is meant as the formal definition for the language, documenting its syntax and semantics, and prescribing minimal functionality for language implementors.</td>
</tr>
<tr>
<td><strong>Pipelines Guide &amp; Reference</strong></td>
<td>The Data Flow oriented companion to NetREXX, with its CMS Pipelines compatible syntax, is documented in this manual. It discusses running Pipes for NetREXX in the command shell and the Workspace, and has ample examples of defining your own stages in NetREXX.</td>
</tr>
</tbody>
</table>
Introduction

A Pipeline, or Hartmann Pipeline\(^\text{1}\), is a concept that extends and improves pipes as they are known from Unix and other operating systems. The name pipe indicates an interprocess communication mechanism, as well as the programming paradigm it has introduced. Compared to Unix pipes, Hartmann Pipelines offer multiple input- and output streams, more complex pipe topologies, and a lot more.

Pipelines were first implemented on VM/CMS, one of IBM’s mainframe operating systems. This version was later adapted to run under MUSIC/SP and TSO/MVS (now z/OS) and has been part of several product configurations. Pipelines are widely used by VM users, in a symbiotic relationship with REXX, the interpreted language that also has its origins on this platform.

Pipes for NetREXX is the implementation of Pipelines for the Java Virtual machine. It is written in NetREXX and pipes and stages can be defined using this language. It can run on every platform that has a JVM (Java Virtual Machine) installed. This portable version of Pipelines was started by Ed Tomlinson in 1997 under the name of njPipes, when NetREXX was still very new, and was open sourced in 2011, soon after the NetREXX translator itself. The included stages have always been open source. It was integrated into the NetREXX translator in 2014 and first released with version 3.04.

In version 3.08, there are important improvements that enable pipelines to be run from the command line, and from the NetREXX REPL program nrws, the NetREXX Workspace. The pipes compiler has been renamed pipc, while the pipes runner component keeps using the name pipe.

\(^1\)\url{https://en.wikipedia.org/wiki/CMS_Pipelines}

\(^2\)This page used to be called Hartmann Pipeline, but was renamed to CMS Pipelines in 2016
The Pipeline Concept

2.1 What is a Pipeline?

The pipeline terminology is a set of metaphores derived from plumbing. Fitting two or more pipe segments together yields a pipeline. Water flows in one direction through the pipeline.

There is a source, which could be a well or a water tower; water is pumped through the pipe into the first segment, then through the other segments until it reaches a tap, and most of it will end up in the sink. A pipeline can be increased in length with more segments of pipe, and this illustrates the modular concept of the pipeline.

When we discuss pipelines in relation to computing we have the same basic structure, but instead of water that passes through the pipeline, data is passed through a series of programs (stages) that act as filters.

Data must come from some place and go to some place. Analogous to the well or the water tower there are device drivers that act as a source of the data, where the tap or the sink represents the place the data is going to, for example to some output device as your terminal window or a file on disk, or a network destination.

Just as water, data in a pipeline flows in one direction, by convention from the left to the right.

2.2 Stage

A program that runs in a pipeline is called a stage. A program can run in more than one place in a pipeline - these occurrences function independent of each other.

The pipeline specification is processed by the pipeline compiler, and it must be contained in a character string; on the commandline, it needs to be between quotes, while when contained in a file, it needs to be between the delimiters of a NetRexx string. An solid vertical bar | is used as stage separator, while other characters can be used as an option when specifying the local option for the pipe, after the pipe name.\(^3\)

When looking a two adjacent segments in a pipeline, we call the left stage the

---

\(^3\)In versions before Pipelines for NetRexx 3.08, the default was the exclamation mark (!), which use was discontinued in favour of conformity with VM/CMS Pipelines.
producer and the stage on the right the consumer, with the stage separator as the connector.

2.3 Device Driver

A device driver reads from a device (for instance a file, the command prompt, a machine console or a network connection) or writes to a device; in some cases it can both read and write. An example of a device drivers are < and >; these read and write data from and to files.

A pipeline can take data from one input device and write it to a different device. Within the pipeline, data can be modified in almost any way imaginable by the programmer.

The simplest process for the pipeline is to read data from the input side and copy it unmodified to the output side. Chapter 7 on page 14 shows the currently supported input- and output devices. The pipeline compiler connects these programs; it uses one program for each device and connects them together.

The inherent characteristic of the pipeline is that any program can be connected to any other program because each obtains data and sends data through a device independent standard interface. This becomes apparent when data can be in-line (specified or generated within the pipeline specification), come in (or be output) to devices like disk or tape, or be handled through a network – all these formats can be processed by the same stages.

The pipeline usually processes one record (or line) at a time. The pipeline reads a record for the input, processes it and sends it to the output. It continues until the input source is drained.
Running pipelines

There are a number of ways to specify and run a pipeline. A little setup is necessary.

3.1 Configuration

The required configuration is minimal. The NetRexxF.jar (java archive file) needs to be on the classpath environment variable (NetRexxC.jar, which is smaller, will suffice when there is a working javac compiler). Also, the current directory (.) needs to be on the classpath. It is convenient to have aliases or shell scripts defined as abbreviations for the invocation of the pipe, pipc (pipe compiler) and nrc (netrexx compiler) utility programs. Aliases are preferable because some shell processors have idiosyncrasies in the treatment of script arguments. With an alias we can be sure that every NetRexx program sees its arguments the same way.

.bash_aliases:
alias pipc="java org.netrexx.njpipes.pipes.compiler"
alias pipe="java org.netrexx.njpipes.pipes.runner"
alias nrc="java org.netrexx.process.NetRexxC"

For Windows, the following works for the pipes runner: file pipe.bat:

@java -cp "%NETREXX_HOME%\lib\NetRexxF.jar;%CLASSPATH%"
   org.netrexx.njpipes.pipes.runner %*

For Windows, the following works for the pipes compiler: file pipc.bat:

@java -cp "%NETREXX_HOME%\lib\NetRexxF.jar;%CLASSPATH%"
   org.netrexx.njpipes.pipes.compiler %*

The previous examples should have the command on one line; the lines are split here because of the limited space on a page. Do note that the Windows .bat files assume that the NETREXX_HOME environment variable is set correctly, that is, to the top of the path where NetRexx is installed. This prepends the NetRexxF.jar file to an already existing CLASSPATH. For the development of local classes (that is, all precompiled pipelines), a dot (’.’), needs to be on this CLASSPATH.

These aliases (or command scripts (in Windows it is called a batch file) enable
you to do the following:
To run a pipeline from the commandline, type:

```
pipe 'gen 100 | dup 999 | count words | console'
```

Remember to use double quotes on Windows shells. When the `pipe` alias or command script is not on your path, you can also use:

```
java org.netrexx.njpipes.pipes.runner 'gen 100 | dup 999 | count words | console'
```

In both cases the answer should be 100000 - you have generated one hundred thousand lines, but fortunately you did not print them, but only counted them. To see them all, you can insert a `| console` stage in between the `dup` and the `count` stage.

After we have verified the working of the command processors, we will discuss in the next section which possibilities you have for running pipelines in day-to-day usage.

### 3.2 From the NetREXX Workspace (nrws) with direct execution

The first way is the most straightforward, and highly recognizable for users of CMS Pipelines, as it mimics the way a pipe is run in the CMS 3270 interface. It also yields the best response time, specially when the `nrws.input` file in your home directory preloads the Pipes subsystem, as in this example:

```
-- preload the pipe machinery for good response on first pipe
pipe literal Pipelines processor loaded | console
```

This is not magic: we do a Pipe execution (that displays: ``Pipe processor loaded``) which loads all necessary classes and leaves them in memory. We can then type this command after the `nrws>` prompt.

![Run in the NetREXX Workspace](image)

```
pipe literal a man a plan a canal panama | reverse | console
```

Executed this way, the executed class image will not be written to disk. The `timing` option is great for prototyping and performance work.

### 3.3 From the command line with direct execution

The only difference is that after the `pipe` command, the rest of the specification needs to be quoted in the command shells of Linux, Windows and macOS. In
CMS, the pipeline specification can also be quoted - in this way, a pipeline can be entirely portable. Windows needs double quotes, zVM/CMS does not need quotes, but if they are used they need to be double quotes. macOS and Linux can use single or double quotes, in most cases.

\texttt{pipe \"literal a man a plan a canal panama | reverse | console\"}

![Figure 2: Run from the OS command line](image)

Executed this way, the executed class image again will not be written to disk.

### 3.4 Precompiled Pipelines

In this mode, which uses the \texttt{pipc} command (for pipe compiler), a .class file will be persisted to disk. This class can be run as many times as needed, without the overhead of compilation. This also would be the right mode for pipes that take different arguments when re-run. The pipe name needs to be specified, and will be the class name. When the class name exists, it will be overwritten.

\texttt{pipc \"(test1) literal a man a plan a canal panama | reverse | console\"}

![Figure 3: Precompile a Pipeline from the OS command line](image)

This will yield a

\texttt{test1.class}

class file, which can be executed by the java virtual machine.

The file \texttt{test1.class} can be run with the command

\texttt{java test1}

Be sure to leave out the .class suffix when invoking java.

### 3.5 Compiled from an .njp file

When compiled from a file, the pipe specification must not be quoted. Pipes can be specified in so-called /emphPortrait Mode, which is the standard for more complex pipelines as it is easier to read. An example is:

\texttt{4 or an appropriate shortcut in modern shells}
pipe (appendtest)

    gen 100  |
    append gen 50  |
    rexx locate /0/  |
    console

3.6 Compiled from an .njp file with additional stage definitions in NetREXX

An example (length1.njp) is:
pipe (lengthp) < output.lst | length1 | console

import org.netrexx.njpipes.pipes.
class length1 extends stage final
method run()
    do
        loop forever
        line = rexx peekto()
        l = line.length
        output(l l.d2x line)
        readto()
    end
    catch StageError
        rc = rc()
    end
    exit(rc*(rc<>12))

In this example, the name of the generated pipe is lengthp, while the name of the custom stage is length1. Be sure to invoke the right class, invoking length1 will have the JVM complain about a non-existing main method. This class (lengthp) will be generated by the command:

    pipc length1

note that the .njp suffix is optional when invoking the pipes compiler. When run, it tries to read the contents of the file length.nrx and will put out its lines, prepended by the line length in decimal and hex - because that is what the (in NetREXX) specified homegrown stage does.
Example Session

Imagine you have landed a job as programmer in an accounting firm, and on your first day there is a question about backups; *the backup process takes too long*. There is an urgent need to identify the files that are produced on this day. You know how to this, of course, it is only some 20 lines of code; use the File() API, fill a collection class (you are thinking of an ArrayList already), or a TreeMap to sort the File object on last modified date already, call an instance of the Calendar class, run a comparison - get that compiled and test it a bit - an hour or so would be sufficient. Of course, you need to install the Java compiler, because all machines have Java nowadays, but just not the compiler. But if you want to really impress people, you should type in a command line and be done with it. For this you can use NetRexx pipelines. Fortunately, you emailed the NetRexxF.jar to yourself so you save it on the machine, and you’re in business right away; you add it to the classpath. Your first pipeline command should just test the waters. For this chapter, we will use the

```
nrws
```

program. You send a command into the pipeline, and get its output:

```
pipe command ls -laFTl | console
```
The `ls` command with the flags is the unix way to get a directory listing - for Windows we would use `dir`. In this case, we send the output into the pipeline, but as the last stage (called a pipe 'sink') occurs immediately after that, every line will be echoed on the console. A number of lines like these will be displayed on the console, as in example 1.

You see straight away that the relevant info is not in the first columns, and not in consecutive columns; we want to know the date (whether it is today or not) and not the time. So we filter this out of every line with a specs stage, as in example 2.

```
pipe command ls -laFTl | specs 42-47 1 58-∗ 8 | console
```

We can easily sort this, with almost no programming:

```
pipe command ls -laFTl | specs 42-47 1 58-∗ 8 | sort | console
```

So what now comes out of the pipeline is sorted (see example 3). But this is a bit funny, we would like to see chronological order of course, so we switch around some columns with another specs stage:

```
pipe command ls -laFTl | specs 42-47 1 58-∗ 8 | specs 7-11 1 1-6 7 12-∗ 12 | sort | console
```

which is very near to what we want (see example 4). Only thing to do now is to filter on the date. We use the `locate` stage and hardcode the date for now. Let's say it is the 2nd of March, 2019:

```
pipe command ls -laFTl | specs 42-47 1 58-∗ 8 | specs 7-11 1 1-6 7 12-∗ 12 | locate /2019 Mar 2/ | sort | console
```

As example 5 shows, on that day there were only two files produced. Also, because this is a short list now, you can see that Pipelines runs this pipe in 0.157
FIGURE 5: example 2

seconds, because we switched on the time option in nrws. Normally, you would specify your pipeline in a file and use portrait mode: commandtest.njp:

pipe (newfiles)
command ls -laFTl |
specs 42-47 1 58-* 8 |
specs 7-11 1 1-6 7 12-* 12 |
sort |
locate /2019 Mar 2/ |
console

The filename is different from the generated class file name, on purpose. You could, and would, put different related pipelines in one file. Then we do a:

pipc commandtest & java newfiles
FIGURE 6: example 3

FIGURE 7: example 4

FIGURE 8: example 5
Write your own Filters

So we have seen in the previous example that it is not too hard to make a simple pipeline out of things called 'device drivers' (such as command, for OS commands, '<' for reading files on disk, and literal, for inserting literal strings into a pipeline, filters, and sinks. When a filter is not delivered in the standard set of stages, it is very easy to make one yourself in the NetREXX language. The model for this closely follows the way it is done with CMS Pipelines and Classic Rexx. Imagine, for the sake of argument (and a simple example\(^5\)), that you have an assignment to quickly reverse a string.

```netrexx
/* BAGVENDT REXX -- Reverse the contents of lines in the pipeline */
signal on error
do forever
    'peekto data'
    'output' reverse(data)
    'readto'
end
error: exit RC*(RC<>12)
```

And you would need to remember to call your filetype REXX instead of EXEC. The peekto reads the input but does not actually commit the read yet, so you can read it one more time with knowledge about the contents. The output pushes its argument back into the pipeline. The readto reads and commits the read so the line is really processed and we can go to the next one.

In NetREXX, that would be about the same, but for some small changes incurred by the object oriented model of NetREXX, which does not exist in Classic Rexx. Here peekto(), readto() and output() are method calls on the stage object. This will be made addressable by the import from org.netrexx.njpipes.pipes. (file: bagvendt.nrx)

```netrexx
import org.netrexx.njpipes.pipes.
class bagvendt extends stage
method run()
    loop forever
        line = Rexx peekto()
        output(line.reverse())
        readto()
    catch StageError
        rc = rc()
    end
exit(rc*(rc<>12))
```

\(^5\)From the document CMS Pipelines Explained, by John P. Hartmann
So that would look fairly familiar, and admittedly, a bit easier for us already well versed in NetRexx. Because the source uses pipe idioms, the regular NetRexx compiler cannot understand everything, and we need to use the pipes compiler `pipc` to compile this source. This will call the NetRexx and Java compilers at the appropriate moment. The resulting .class file needs to be on the CLASSPATH environment variable.

We can test this by building a pipeline and running the filter on its own source:

```ruby
pipe "literal abcd | bagvendt | console"
```

If you have a CMS handy, that would be:

```ruby
pipe literal abcd | bagvendt | console
```

on the first, Classic Rexx version of the filter - but the quoted version also works on CMS.

![Figure 9: BAGVENDT under VM/CMS](image)
FIGURE 10: bagvendt.nrx under NetRexx
More advanced Pipelines

Admittedly, the examples in the previous chapters could have been done with Unix pipes or at least with incorporation of stream utilities like awk or sed.

To get a good idea of what can be done with Pipelines for NetRExx, look at the tasktest pipe in the examples directory. It implements the shell of a multitasking server - using about eight stages. The file examples/tcptask.njp contains an example of this technique being used.

```bash
pipe (tasktest stall 2000 -gen)

literal 0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S
T |
dup 2 |
split |
                      -- supply work for task stage
ptimer |
a: deal secondary ?
        -- send work to task stage requesting
        work
b: faninany |
        -- buffer requests to so no deadlocks
        elastic |
        ptimer |

a: |
    copy |
    task 1 |
    -- worker task 1
b: ?

a: |
    copy |
    task 2 |
    -- worker task 2...
b: ?

a: |
    copy |
    task 3 |
b: 

Before discussing this example in-depth, we need to go into some more basic concepts.

\(^6\) using code from Melinda Varians ‘Cramming for the Journeyman Plumber Exam’ paper
Device Drivers

Pipelines for NetRexx contains the following device drivers:
(see next page)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>read from a file</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>write to a file (which is overwritten if it exists)</td>
</tr>
<tr>
<td><code>»</code></td>
<td>append to a file (which is created if it does not exist)</td>
</tr>
<tr>
<td><code>diskr</code></td>
<td>read from a file</td>
</tr>
<tr>
<td><code>diskw</code></td>
<td>write to a file (which is overwritten if it exists)</td>
</tr>
<tr>
<td><code>diska</code></td>
<td>append to a file (which is created if it does not exist)</td>
</tr>
<tr>
<td><code>diskslow</code></td>
<td>read, create or append to a file</td>
</tr>
<tr>
<td><code>array</code></td>
<td>manipulate arrays</td>
</tr>
<tr>
<td><code>arraya</code></td>
<td>manipulate arrays</td>
</tr>
<tr>
<td><code>arrayr</code></td>
<td>manipulate arrays</td>
</tr>
<tr>
<td><code>stem</code></td>
<td>manipulate stems</td>
</tr>
<tr>
<td><code>stema</code></td>
<td>manipulate stems</td>
</tr>
<tr>
<td><code>stemr</code></td>
<td>manipulate stems</td>
</tr>
<tr>
<td><code>vector</code></td>
<td>manipulate vectors</td>
</tr>
<tr>
<td><code>vectora</code></td>
<td>manipulate vectors</td>
</tr>
<tr>
<td><code>vectorr</code></td>
<td>manipulate vectors</td>
</tr>
<tr>
<td><code>var</code></td>
<td>read or set a variable in a NetRexx program</td>
</tr>
<tr>
<td><code>zip</code></td>
<td>compress a set of files (0 or more) into a zip archive</td>
</tr>
<tr>
<td><code>unzip</code></td>
<td>decompress a set of files (0 or more) from a zip archive</td>
</tr>
<tr>
<td><code>listzip</code></td>
<td>list a zip file directory</td>
</tr>
<tr>
<td><code>console</code></td>
<td>read from, or write to a terminal (window)</td>
</tr>
<tr>
<td><code>hole</code></td>
<td>destroy data</td>
</tr>
<tr>
<td><code>delay</code></td>
<td>suspend stream</td>
</tr>
<tr>
<td><code>literal</code></td>
<td>write the argument string</td>
</tr>
<tr>
<td><code>strliteral</code></td>
<td>write the argument string</td>
</tr>
<tr>
<td><code>sqlselect</code></td>
<td>select from any jdbc source</td>
</tr>
<tr>
<td><code>xrange</code></td>
<td>write a character range</td>
</tr>
</tbody>
</table>
Record Selection

Various stages can select records and work on data in the pipeline. These are stages called select, sort, specs, locate, etcetera. For a complete description we refer to the IBM Pipelines documentation.

These are the main selection stages supported in Pipelines for NetRexx:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>between</td>
<td>selects records between labels</td>
</tr>
<tr>
<td>drop</td>
<td>discard records from the beginning or the end of a file</td>
</tr>
<tr>
<td>find</td>
<td>select lines</td>
</tr>
<tr>
<td>strfind</td>
<td>select lines</td>
</tr>
<tr>
<td>frrlabel</td>
<td>select records from the first one with leading string</td>
</tr>
<tr>
<td>strfrrlabel</td>
<td>select records from the first one with leading string</td>
</tr>
<tr>
<td>inside</td>
<td>select records between labels</td>
</tr>
<tr>
<td>locate</td>
<td>select records between labels</td>
</tr>
<tr>
<td>nfind</td>
<td>select lines using xedit nfind logic</td>
</tr>
<tr>
<td>strnfind</td>
<td>select lines using xedit nfind logic</td>
</tr>
<tr>
<td>nlocate</td>
<td>select lines without a string</td>
</tr>
<tr>
<td>notinside</td>
<td>select records not between labels</td>
</tr>
<tr>
<td>outside</td>
<td>select records not between labels</td>
</tr>
<tr>
<td>pick</td>
<td>select records that satisfy a relation</td>
</tr>
<tr>
<td>take</td>
<td>select records from the beginning or the end of a file</td>
</tr>
<tr>
<td>tolabel</td>
<td>select records to the first one with leading string</td>
</tr>
<tr>
<td>strtolabel</td>
<td>select records to the first one with leading string</td>
</tr>
<tr>
<td>unique</td>
<td>discard or retain duplicate lines</td>
</tr>
</tbody>
</table>
## Filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>buffer records</td>
</tr>
<tr>
<td>chop</td>
<td>truncate the record</td>
</tr>
<tr>
<td>join</td>
<td>join records</td>
</tr>
<tr>
<td>pad</td>
<td>expand short records</td>
</tr>
<tr>
<td>split</td>
<td>split records relative to a target</td>
</tr>
<tr>
<td>change</td>
<td>substitute contents of records</td>
</tr>
<tr>
<td>specs</td>
<td>rearrange contents of records</td>
</tr>
<tr>
<td>xlate</td>
<td>transliterate contents of records</td>
</tr>
<tr>
<td>copy</td>
<td>copy records</td>
</tr>
<tr>
<td>count</td>
<td>count lines, words and bytes</td>
</tr>
<tr>
<td>dup</td>
<td>duplicate the object</td>
</tr>
<tr>
<td>reverse</td>
<td>reverse contents of records</td>
</tr>
<tr>
<td>timestamp</td>
<td>prefix date and time to records</td>
</tr>
<tr>
<td>append</td>
<td>put output from device driver after data on the primary input</td>
</tr>
<tr>
<td>casei</td>
<td>run selection stage in a case-insensitive manner</td>
</tr>
<tr>
<td>not</td>
<td>run stages with output streams inverted</td>
</tr>
<tr>
<td>prefix</td>
<td>Blocks its primary input and executes stage supplied as an argument</td>
</tr>
<tr>
<td>zone</td>
<td>run selection stage on subset of input record</td>
</tr>
<tr>
<td>elastic</td>
<td>buffer sufficient records to prevent stall</td>
</tr>
<tr>
<td>fanin</td>
<td>concatenate streams</td>
</tr>
<tr>
<td>faninany</td>
<td>copy records from whichever input stream has one</td>
</tr>
<tr>
<td>gate</td>
<td>pass records until stopped</td>
</tr>
<tr>
<td>juxtapose</td>
<td>preface record with marker</td>
</tr>
<tr>
<td>overlay</td>
<td>overlay data from input streams</td>
</tr>
<tr>
<td>command</td>
<td>issue a command and write response to pipeline</td>
</tr>
</tbody>
</table>
Other Stages

<table>
<thead>
<tr>
<th>query</th>
<th>check version and level of Pipelines for NetRexx</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- --</td>
<td>insert comments into a pipeline</td>
</tr>
<tr>
<td>comment</td>
<td>insert comments into a pipeline</td>
</tr>
</tbody>
</table>
Multi-Stream Pipelines

One of the defining differences with Unix pipes is the possibility to define multi-stream pipelines. The selection stages from the previous chapter all have secondary streams. What the selection parameters have discarded, seem to have discarded, is in reality not gone. In fact, Pipelines for NetRexx throws very little away during execution.

The way to use the not-selected part of the data through these secondary streams is explained in this chapter; it is this capacity that constitutes the freedom to work with many different streams in one pipeline; where Unix pipes are limited to not very much more than stdin, stdout, stderr -- Pipelines for NetRexx enables the user to define as many streams as necessary to accomplish the task at hand in an efficient manner.

Let us look at a simple selection like the following:

```
pipe literal foo bar baz frob frobnitz frobbotzim | split | locate /oo/ |
    console
```

foo

The string that makes it through the selection that is done by the `locate` is 'foo' - it is the only one that is captured by the `/oo/` filter.

The rest of the words is not gone, however, and we can use these in further processing by using the secondary stream that `locate` provides.

To prepare for this, we give the secondary stream a name by providing a label for it, we call it, in absence of any creativity, `rest`\(^\text{\textsuperscript{2}}\). Also, we send the selected output, `foo` into a `hole` stage, where it disappears.

```
pipe literal foo bar baz frob frobnitz frobbotzim | split | rest: locate /oo/ |
    hole
```

As predicted, there is no output. To get to the rest of the words, unselected by `locate`, we connect the secondary output stream to a new pipe, using the '?' (the default pipe-end character) like this:

```
pipe literal foo bar baz frob frobnitz frobbotzim | split | rest: locate /oo/ |
    hole ? rest: | console
```

The output is now:

\(^{2}\text{\textsuperscript{2}}\)often, you will see it being called 'a:'
bar
baz
frob
frobnitz
frobbotzim

Instead of sending the original output into a black hole, we could have also gone further with it, and, for example, reverse it:

```
pipe literal foo bar baz frob frobnitz frobbotzim | split | rest:
    locate /oo/ |
    reverse | console | rest: | console
```

The output is now:

```
oof
bar
baz
frob
frobnitz
frobbotzim
```

Likewise, we can specify more filter stages in the second, attached pipeline, and bifurcate the pipeline even further.

```
pipe literal foo bar baz frob frobnitz frobbotzim | split | rest:
    locate /oo/ |
    reverse | console | rest: | locate /botzim/ | console
```

The output is now:

```
oof
frobbotzim
```

It is good to define and implement secondary streams when you write your own stages.
Pipeline Stalls

With multistream pipelines a new problem sometimes rears its head - a Pipeline stall, also called deadlock. This happens when stages wait for input that cannot be delivered, in a way that ensures that it cannot be delivered.

Pipes for NetRexx detects deadlocks and outputs information to allow you to fix the problem. Consider the following session:

```
pipe literal test | a: fanin | console | a:
```

![Figure 11: Deadlock detection](image)

We can see that there are three stages in the Running state. None have any return codes set. The Flags tell us that all the stages are waiting for an output to complete.

The ‘->’ show which stream is selected. From this we can see console_3 is trying to output to fanin_2. Unfortunately fanin_2 is waiting for output on stream 0 to complete, it cannot read the data waiting on in stream 1. Hence the stall.

The strings after Dumping and Monitored by are the autogenerated class names. When you name your pipelines with precompiled pipes yourself, the names you have given them will be displayed here.

When a stream has data being output, there is a boolean flag following the name
of the stage the stream is connected to. This tracks the peek state of the object. For an output stream, true means the following stage has peeked at the value. With input streams, the current stage has seen the value when its true.

When a stage is multithreaded, like elastic, you can get flags of 3 or 5. This means that threads are waiting on output and read, or output and any. When using multithreaded stages, only one thread should use output unless it is serialized using protected or synchronized blocks.

When a stage has a pending sever or autocommit, flag bits are set too.
How to use a pipe in a NetRexx program

This shows how to use a pipe in a NetRexx program:

```rexx
class testpipe

    method testpipe(avar=Rexx)
        F = Rexx 'abase'
        T = Rexx 1

        F[0]=5
        F[1]=222
        F[2]=3333
        F[3]=1111
        F[4]=55
        F[5]=444

        pipe (apipe stall 1000 )
        stem F | sort | prefix literal {avar} | console | stem T

        loop i=1 to T[0]
            say 'T['i']=T[i]
        end

    method main(a=String[]) static
        testpipe(Rexx(a))

```

A couple of things can be seen in this example. First that it is simple to pass NetRexx variables to pipes using `stem`. Also look at the phrase `{avar}`. It passes the NetRexx variable's value to the stage at runtime. In CMS the pipe would be quoted and you would unquote sections to get a similar effect.

Another thing to note is that the pipe extraction program is fairly smart. It detects when pipes takes several lines. As long as there are stages, or the current line ends with a stagesep or stageend character, or the next line starts with a stagesep or stageend character. It gets added to the pipe.

The `arg()`, `arg(rexx)` or `arg(null)` methods get the arguments passed to a stage or pipe. To get the complete rexx string of an argument use `arg()`. To get the nth word of a rexx argument use `arg(n)`. When using pipes in netrexx code you can use `arg('name')` to get the named argument. If the class of the argument is not rexx use `arg(null)` to get the object.

In .njp files you can use `avar` phrase actually just shorthand for `arg('avar')`. The following example shows what has to be done in a stage to access the rexx variables passed by VAR, STEM and OVER. The real over stage is a bit more com-
The `getRexx` method is passed the name of a string by the pipe. In the previous example it would be passed `A` and would return an Object pointer to `A` in `testpipe`. If you wish to replace a stream this can be done using connectors. For example look at the following fragment:

```java
-- examples\calltest.njp
pipe (callt1) literal test | calltest {} | console
import org.netrexx.njpipes.pipes.

class calltest extends stage final
method run() public
    do
        a = arg()
        callpipe (cp1) gen {a} | *out0:
            loop forever
                line = peekto()
                output(line)
                readto()
            end
        catch StageError
            rc = rc()
        end
    end
exit(rc*(rc<>12))
```

Running the `callt1` pipe with an argument of 10 would pass the 10 to `calltest` via and `arg()`. Then `cp1`'s `gen` stage would be passed 'a' which is set to 10. Since `gen` generate numbers in sequence, the console stage of `callt1` would get the numbers from 1 to 10. Now `cp1` ends and `calltest`’s output stream is restored and `calltest` unblocks and reads the the literal’s data 'test' and passes it to `console`.

The use of only works when compiling from `.njp` files. It will not work from the command line. The `njpipes` compiler recognizes connectors as labels with the following forms:

*in:
When N is a whole number, the connector connects input or output stream N of
the stage with the connector. When the label *in or *out, the connector connects
the stage's current input or output stream with the connector. This is used instead of *: due to the way the compiler/preprocessor works. If you do not want
the stage to wait for the called pipe to complete you can use addpipe. Here is
an example.

```
-- similar to examples\addtest.njp

a = 100
b = 'some text for literal'

addpipe (linktest) literal {b} | dup {a} | *in0:
    loop forever
        line = Rexx readto()
    catch StageError
    end

readto() will get 'some text for literal' one hundred times.

A quick aside. When writing stages remember that njPipes moves objects
through pipes. Use 'value = peekto()' instead of 'value = rexx peekto()' when
ever possible. Some of the supplied stages pass objects with classes other than
rexx and forcing rexx will cause classCastExceptions. If a stage needs a rexx
object try using the rexx stage modifier to attempt to convert the object. Feel
free to expand this stage, but please send me the updated version.

Serious stage writers will probably want to take a good look at the methods
defined in the NetRExx source package org.netrexx.process.njpipes.stages.
There you will find various methods for parsing ranges. You will also find the
stub for the stageExit compiler exit. It can be used to produce 'on the fly' code at
compile time. You can also use it to change the topology of the unprocessed part
of the pipe. The major use is to allow implementations of stages like prefix, appen
d or zone. Its also used to produce better performing stages, for an example
see specs. The compiler also queries the rexxArg() and stageArg() methods. If
your stage expects objects of class Rexx as arguments rexxArg() should return
the number of variables expected. If your stage expects a stage for an argument,
stageArg() should return the word position of the stage.
Giving commands to the operating system

The command stage is used to issue commands to the operating system and trap the output to the pipeline. command can receive its input as parameters, or through the pipeline. So

```
pipe literal ls | command | sort | console
```

is equivalent to:

```
pipe command ls | sort | console
```

14.1 Built-ins

Some commands, like dir in Windows, do not have a separate executable file; there is no dir.exe. This can be solved by having the command processor, cmd.exe start its built-in command. The pipeline would be, for example:

```
pipe literal cmd /c dir | command | sort | console
```
TCP/IP Networking

As the built-in stages all work on data that is dispatched through the pipeline, irrespective of which device driver is used, it is also convenient to do network programming using a set of pipelines.

The `tcplisten` stage can be used as a network device driver, as in CMS, but limited to specification of the port and a timeout value. Below an example of how to implement a sample TCP/IP client/server application.

```plaintext
-- one shot tcpip server
pipe (tcpserv stall 60000 debug 0 )
   tcplisten 1958 timeout 15000 | tcpexample

-- one shot tcpip requestor
pipe (tcpreq stall 60000 debug 0 )
   random {} |
   specs ** 1 ,\n, next |
   tcpclient deblock c localhost 1958 timeout 10000 linger 500
       oneresponse |
   rexx to console

-- a single tasking server
options binary
import org.netrexx.njpipes.pipes.
class tcpexample extends stage
method run() public
   loop forever
      peekto()
      callpipe (tcplog stall 15000 debug 0)
         *in0: |
         take first 1 |
         console |
      f: fanin |
         tcpdata timeout 10000 deblock C oneresponse |
         elastic |
         insert /\n/ after |
      f:
      catch StageError
         rc = rc()
   end
```
This example needs to be compiled with the pipes compiler, see TCP/IP Client/Server compile, which yields the classes tcperv and tcpreq, for the server and the requester component.

Now we can start the generated pipelines each in their own shell window. As can be seen in TCP/IP server, the class keeps waiting on connections on port 1958 - which is arbitrary, but specified in the pipeline source.

In another window, we can start the TCP/IP requestor, which when given port 1958 as argument, connects to the server, and displays a series of random numbers that is sent to it.

Note that the stage tcpexample from the tcpserver pipeline is a custom stage that is written in this tcpexample.njp file.
Selecting from relational databases

Using the built-in sqlselect stage you can select data, using SQL, from any jdbc source available.

An sqlselect.properties file is needed to define the jdbc parameters like the driver to use, the url of the data source and other arguments, like a password and tracing options, if needed.

The file looks like this:

jdbcdriver=org.sqlite.JDBC
url=jdbc:sqlite:flightroute-iata.sqb

This is all that is needed for an sqlite database containing flight data. A simple select * can then be done with the following pipeline:

pipe literal ∗ from FlightRoute where flight = 'KLM765' | sqlselect | console

This yields the following output:

FLIGHT--ROUTE--UPDATETIME--
KLM765  AUA-BON-AMS  1494132448

Note that from the command line, the quotes around the pipe specification and the literal string in the SQL statement should be opposite, while when the pipeline is issued from the Workspace for NetREXX, the pipeline does not have to be quoted, but the sql string needs double quotes instead of the - for SQL statements- normal single quotes.
The Pipes Runner

The pipes compiler is used in both precompiled and directly executed pipelines. When you directly execute a pipeline from the command line or from the nrws NetREXX workspace, the process is optimized to not persist generated NetREXX, Java and Class files to disk before execution; the whole process runs from memory. The Pipes Runner uses the Pipes Compiler for this purpose, and as such misses the options for persistence.

The pipe command alias start the Pipes Runner, which is a command processor that can execute a pipe from the command line in an OS shell, the OS being Windows, Linux or macOS.

A pipe can be run with options prepended within parentheses, like this:

pipe '(test1 sep ! stall 2000 debug 63) literal abcde ! console'

The following options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Specify the name of the generated class file. This can be useful for debug</td>
</tr>
<tr>
<td></td>
<td>ping purposes but is not mandatory when running a pipe. An unnamed pipe</td>
</tr>
<tr>
<td></td>
<td>receives a generated unique name. This option needs to go first.</td>
</tr>
<tr>
<td>sep</td>
<td>The default stage separator is the</td>
</tr>
<tr>
<td>debug</td>
<td>The debug option specifies a bitmask for debugging the execution of a pipe;</td>
</tr>
<tr>
<td></td>
<td>(debug 63), for example, generates a rather complete debugging trail.</td>
</tr>
<tr>
<td>end</td>
<td>The default pipe end character is the ’?’ (question mark), which can be</td>
</tr>
<tr>
<td></td>
<td>overridden here. Note that the backslash, which is an obvious pipe end</td>
</tr>
<tr>
<td></td>
<td>character for the z/VM 3270 interface, is not a good choice for Windows</td>
</tr>
<tr>
<td></td>
<td>and Unix shells.</td>
</tr>
<tr>
<td>stall</td>
<td>The duration in number of seconds of a pipe stall (or deadlock) detection</td>
</tr>
<tr>
<td></td>
<td>cycle.</td>
</tr>
</tbody>
</table>

---

8But specifying them will not generate an error
9this is a non-exhaustive list of operating systems
The Pipes Compiler

The purpose of precompiling a pipeline specification is to produce a .class file for the JVM that can be run independently and on different machines; only the JVM and the NetRexxC.jar or the NetRexxF.jar are required to run a precompiled pipe. A set of precompiled pipes can be shipped as an application.

When precompiling pipes, there are options to save and view the generated NetRexx, Java and JVM Class files. A precompiled pipe has the advantage that it can be executed over and over in an application, without the need to compile it every time; the performance savings are accumulative in this scenario.

The following options can be used on the pipc command, in addition to the ones specified in the previous chapter for the Pipes Runner:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-gen</strong></td>
<td>Generate the NetRexx source file. The pipeline needs a name.</td>
</tr>
<tr>
<td><strong>-keep</strong></td>
<td>Keep the Java source which is generated from the NetRexx source.</td>
</tr>
</tbody>
</table>

Example:

pipe (testpipe -gen -keep)

This will generate the NetRexx source as well as keep the java source.
Built-in Stages

This section describes the set of built-in stages, i.e. the ones that are delivered with the downloadable open source package. These stages are directly executable from the NetRexxC.jar file or the NetRexxF.jar file (the latter contains a Java compiler for use on JRE-only systems); also, the source of these stages is delivered in the NetRexx source repository. This repository can be checked out at

```
git clone https://git.code.sf.net/p/netrexx/code netrexx-code
```

The source of the stages is in directory

```
netrexx-code/src/org/netrexx/njpipes/stages
```
Differences with CMS Pipelines

The goal of this implementation is to be as close as possible to the CMS version of Pipelines. A few differences are unavoidable.

- The character set is Unicode and not EBCDIC, as Unicode is the character set of the underlying Java platform
- As shells are different, many 3270 related stages are not implemented
- Pipes need to be quoted on the Windows and Unix command lines; the Workspace for NetRexx ($nrws$) environment is an exception to this rule
- The mainframe is record-oriented in many stages, Pipelines for NetRexx does an approximation of this
- Pipelines on the mainframe is an interpreted language with components as the scanner and the dispatcher; the NetRexx version is compiled to Java .class files by $pipc$, the pipes compiler, and dispatched as threads by the JVM.
- The mainframe pipes dispatcher is not multiprocessor enabled. In Pipelines for NetRexx all tasks (stages) are dispatched over all available processors in parallel.
- The fact that pipes run from NetRexx implies that they can be used in Java source. In previous releases there was more direct support for this; this has lapsed due to changes in the way a java toolchain works. This support can be restored in future releases.
- To put the content of a NetRexx variable in a pipe specification in a NetRexx program, there is a $\{\}$ mechanism. In CMS the pipe would be quoted in the Rexx source and you would unquote sections to get a similar effect.
### Stages Built Into
NetRexx Pipelines 4.02
&
CMS Pipelines V7R1
and Their Differences

#### How to Read Syntax Diagrams

Special diagrams (often called railroad tracks) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The ▶▶ symbol indicates the beginning of the syntax diagram.
- The --- symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The ▶ symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The --- symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults.

#### Pipelines Built In Stages

<table>
<thead>
<tr>
<th>Show Stages Implemented in:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NetRexx Pipelines:</td>
<td>CMS Pipelines:</td>
</tr>
</tbody>
</table>

Show All Details: (Double click on a row to turn it on/off.)

Highlight NetRexx Only / CMS Only:

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
<th>syntax diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;diskw filew 3.09</td>
<td>Replace or Create a File</td>
<td>▶▶---&gt;--string--◄</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* delegates to diskw.</td>
</tr>
<tr>
<td>&gt;&gt;diska filea 3.09</td>
<td>Append to or Create a File</td>
<td>▶▶---&gt;--string--◄</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* delegates to diska.</td>
</tr>
<tr>
<td>&gt;&gt;mvs</td>
<td>Append to a Physical Sequential Data Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>&gt;&gt;oe</td>
<td>Append to or Create an OpenExtensions Text File</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>&gt;&gt;sfs</td>
<td>Append to or Create an SFS File</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>&gt;&gt;sfsslow</td>
<td>Append to or Create an SFS File</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>&gt;mdsk</td>
<td>Replace or Create a CMS File on a Mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>mvs</td>
<td>Rewrite a Physical Sequential Data Set or a Member of a Partitioned Data Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>oe</td>
<td>Replace or Create an OpenExtensions Text File</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>sfs</td>
<td>Replace or Create an SFS File</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>Read a File</td>
<td></td>
</tr>
<tr>
<td>disk</td>
<td>• Implemented as in CMS; delegates to diskr.</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msk</td>
<td>Read a CMS File from a Mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>mys</td>
<td>Read a Physical Sequential Data Set or a Member of a Partitioned Data Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>oe</td>
<td>Read an OpenExtensions Text File</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>sfs</td>
<td>Read an SFS File</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>sfsslow</td>
<td>Read an SFS File</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td>Comment Stage, No Operation</td>
<td></td>
</tr>
<tr>
<td>comment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3277bfra</td>
<td>Convert a 3270 Buffer Address Between Representations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
<tr>
<td>3277enc</td>
<td>Write the 3277 6-bit Encoding Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
<td></td>
</tr>
</tbody>
</table>
64decode
decode64
3.11

Decode Base-64 Format

- NOTE: CMS is only 64DECODE, and does not have the options; it does MIME.
- BASIC - Output is mapped to a set of characters lying in A-Za-z0-9+/.
- The encoder does not add any line feed in output the decoder rejects any character other than A-Za-z0-9+/.
- URL - Output is mapped to set of characters lying in A-Za-z0-9_.
- Output is URL and filename safe.
- MIME - Output is mapped to MIME friendly format. Output is represented in lines of no more than 76 characters each uses a carriage return 'v' followed by a linefeed 'n' as the line separator. No line separator is present to the end of the encoded output.

64encode
encode64
3.11

Encode to Base-64 Format

- NOTE: CMS is only 64DECODE, and does not have the options; it does MIME.
- BASIC - Output is mapped to a set of characters lying in A-Za-z0-9+/.
- The encoder does not add any line feed in output the decoder rejects any character other than A-Za-z0-9+/.
- URL - Output is mapped to set of characters lying in A-Za-z0-9_.
- Output is URL and filename safe.
- MIME - Output is mapped to MIME friendly format. Output is represented in lines of no more than 76 characters each uses a carriage return 'v' followed by a linefeed 'n' as the line separator. No line separator is present to the end of the encoded output.

? help

Display Help for Pipelines

- Not implemented in Netrexx Pipelines.

abbreviation
abbreviatio
abbreviati
abbreviat
abbrevi
abbrev

Select Records that Contain an Abbreviation of a Word in the First Positions

- (1) ABBREViation must be ABBREV in CMS
- (2) ANYcase must be ANYCASE in CMS

acigroup

Write ACI Group for Users

- Not implemented in Netrexx Pipelines.

addrdw

Prefix Record Descriptor Word to Records

- Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Not Implemented in Netrexx Pipelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>adrspace</td>
<td>Manage Address Spaces</td>
<td>Yes</td>
</tr>
<tr>
<td>aftfst</td>
<td>Write Information about Open Files</td>
<td>Yes</td>
</tr>
<tr>
<td>aggrc</td>
<td>Compute Aggregate Return Code</td>
<td>Yes</td>
</tr>
<tr>
<td>all</td>
<td>Select Lines Containing Strings (or Not)</td>
<td>Yes</td>
</tr>
<tr>
<td>alserv</td>
<td>Manage the Virtual Machine's Access List</td>
<td>Yes</td>
</tr>
<tr>
<td>apldecode</td>
<td>Process Graphic Escape Sequences, Old APL language</td>
<td>Yes</td>
</tr>
<tr>
<td>aplencode</td>
<td>Generate Graphic Escape Sequences, Old APL language</td>
<td>Yes</td>
</tr>
<tr>
<td>append</td>
<td>Put Output from a Device Driver after Data on the Primary Input Stream</td>
<td>Yes</td>
</tr>
<tr>
<td>array</td>
<td>Read or Write an Array</td>
<td>Yes</td>
</tr>
<tr>
<td>arraya</td>
<td>Read or Write an Array</td>
<td>Yes</td>
</tr>
<tr>
<td>arrayr</td>
<td>Read or Write an Array</td>
<td>Yes</td>
</tr>
<tr>
<td>arrayw</td>
<td>Read or Write an Array</td>
<td>Yes</td>
</tr>
<tr>
<td>asatocmc</td>
<td>Convert ASA Carriage Control to CCW Operation Codes. Old printer control</td>
<td>Yes</td>
</tr>
<tr>
<td>asmcont</td>
<td>Join Multiline Assembler Statements</td>
<td>Yes</td>
</tr>
<tr>
<td>asmfind</td>
<td>Select Statements from an Assembler File as XEDIT Find</td>
<td>Yes</td>
</tr>
<tr>
<td>asmnfind</td>
<td>Select Statements from an Assembler File as XEDIT NFind</td>
<td>Yes</td>
</tr>
<tr>
<td>asmxpnd</td>
<td>Expand Joined Assembler Statements</td>
<td>Yes</td>
</tr>
<tr>
<td>beat</td>
<td>Mark when Records Do not Arrive within Interval</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Select Records Between Labels

<table>
<thead>
<tr>
<th>BETWEEN</th>
<th>case</th>
<th>delimitedString1</th>
<th>number</th>
<th>delimitedString2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Xhexstring1</td>
<td></td>
<td>Xhexstring2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hhexstring1</td>
<td></td>
<td>Hhexstring2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bbinstring1</td>
<td></td>
<td>Bbinstring2</td>
</tr>
</tbody>
</table>

case:

<table>
<thead>
<tr>
<th>ANYcase</th>
<th>CASEANY</th>
<th>CASEIGNORE</th>
<th>IGNORECASE</th>
<th>CASELESS</th>
</tr>
</thead>
</table>

Block to an External Format

- Not implemented in Netrexx Pipelines.

Display Data on a 3270 Terminal

- Not implemented in Netrexx Pipelines.

Buffer Records

<table>
<thead>
<tr>
<th>BUFFER</th>
<th>number</th>
<th>delimitedString</th>
</tr>
</thead>
</table>

Combine Overstruck Characters to Single Code Point. Old printer.

Run Selection Stage in Case Insensitive Manner

- (1) CMS Pipelines only.
Substitute Contents of Records

```plaintext
| change |
+-----------------------------+-----------------------------+-----------------------------+-----------------------------|
| Substitute Contents of Records |
| NetRexx Pipelines only. |
```

(1) NetRexx Pipelines only.
**changeparse**

Change the contents of records, using Rexx Parse. Calculations can be done.

```
-NetRexx-
  +--FROM++--TO++
  |+-CHANGEParse--+-parse_template_Dstring--+--output_template_Dstring--|
  |+-BY-by_NetRexx_Dstring++
  |+-FIRST-first_NetRexx_Dstring++
```

- Records are parsed via the `parse_template_delimited_string`.
- Variables are named `$n`, where `n` is 1 to 9.
- The `by_NetRexx_delimited_string` is interpreted. This is 0 or more semicolon separated NetRexx statements, probably using the `$n` variables, which can have the value altered.
- Other variables may be used, and are persistent while the stage is active, so can be used as accumulators.
- The values of the variables are put into the `output_template_delimited_string` replacing `$n`.
- For a literal `$n` that won't be changed, use `$$n`.
- A first `NetRexx_delimited_string`, if present, is interpreted before reading any record from the primary input stream.
- Any variables used in the `by_NetRexx_delimited_string` must be defined here.
- A finally `template_delimited_string`, if present, is written as a final output record after the primary input stream is finished using the `$n`s.
- Any keyword phrases must, in any order, follow any non-keyworded FROM & TO phrases.
- This is NetRexx Pipelines only, not in CMS.

Examples:

```
  * changeparse / 2 $1 +1/ /The second letter is "$1". $1 will be changed./
  * changeparse from / 2 $1 +1/ to /The second letter is "$1". $1 won't be changed./
  * changeparse from / . $2 . 50 $5 +5/ /The product is $1/ by /$1 = $2 * $5/,
    to /The product is $1/,
    by /$1 = $2 * $5; $3 = $3 + $1/,
    first /$3 = 0/,
    finally /$3 is the total/
```

**changeregex**

Substitute Contents of Records using Java Regular Expressions

```
-NetRexx-
  +--FROM++--TO++ +--ONE++
  |+-CHANGERegex--+-delimitedString-(1)+--delimitedString-(2)+--|
  |+-ALL++
```

- Uses the Java RegEx classes and its dialect of RegEx. See Java’s `Pattern` class and `replaceFirst` and `replaceAll` of `String` for full documentation.
- (1) First, FROM, delimitedString is a Java `RegEx expression` for what is to be replaced.
- (2) Second, TO, delimitedString is the replacement string. It may contain elements from the first one.
- This is NetRexx Pipelines only, not in CMS.

**chop**

Truncate the Record

```
  +--80------------+
  |+-CHOP---+-stringtarget-+-|
  |+-TRUNCate++-snumber-------|
  |+-| stringtarget |---+
```

- `stringtarget:`
  - `|-------| target |---|
  - `+-ANYcase-+-+-BEFORE-+ |+-NOT-+-`
  - `+-CASEANY-+-+-+------+-+++`
  - `+-CASEIGNORE-+-snumber-+-AFTER-+-`
  - `+-IGNORECASE-+-`
  - `+-CASELESS-----`

- `target:`
  - `|-------| xrange-+-delimitedString-+-`
  - `+-ANYof--+-`

**cipher**

Encrypt and Decrypt Using a Block Cipher

- Not implemented in Netrexx Pipelines.
### ckddeblok

**Deblock Track Data Record**

- Not implemented in Netrexx Pipelines.

### cmd

**Issue OS Commands, Write Response to Pipeline**

- input stream 0 is for commands
- input stream 1 is stdin
- output stream 0 is stdout
- output stream 1 is the return code
- output stream 2 is stderr

### cms

**Issue CMS Commands, Write Response to Pipeline**

- Not implemented in Netrexx Pipelines.

### collate

**Collate Streams**

- (1) NetRexx Pipelines only.
- (2) delimitedString record is put before each Master Record (or after if DETAIL MASTER order) on the primary output stream.
- 3.11 New to NetRexx Pipelines. Add SEparator option.

### combine

**Combine Data from a Run of Records**

- (1) Only for use with secondary input streams. Only options from this column usable with any secondary input stream. (This is poorly documented in CMS Pipelines. This is a best guess of their intentions.)
- (2) Not usable with STOP and secondary streams.
**command**

Issue OS Commands, Write Response to Pipeline

- input stream 0 is for commands
- input stream 1 is stdin
- output stream 0 is stdout
- output stream 1 is the return code
- output stream 2 is stderr

**comment**

Comment stage

Not in CMS Pipelines;

This is a STAGE, not a programming comment. It must have a SPACE after --.

It must have either a stageEnd or pipeEnd.

If ended with a stageEnd, it passes records through on primary input to output streams.

If ended with a pipeEnd, it does NOT pass records through.

If used before a driver stage, it must have a pipeEnd.

**compare**

Compare Primary and Secondary Streams, Write the Result

- (1) -1 = Primary is shorter/less, 0 = equal, 1 = Secondary is shorter/less
- (2) 0 = equal, 1 = not equal
- (3) Primary is LESS/shorter (or MORE/longer) than secondary
- (4) DStrings can use any of the following escapes (or the lowercase) for the unequal situation:
  - \C (count) for the record number,
  - \B (byte) for column number
  - \P (primary) for the primary stream record
  - \S (secondary) for the secondary stream record
  - \L (Least) for the stream number that is shorter, -1 if equal
  - \M (Most) for the stream number that is longer, -1 if equal
- (5) Equal or not, this DString precedes any of the others.
- (6) This is NetRexx Pipelines only, not included in CMS
- (7) In reporting \P & \S, control characters, except new line, \n, are transliterated to [blob, 219.d2c()]
- (8) Without ECHO, this stops and reports at first non-compare. With ECHO, each primary input is reported; after first compare primary input stream records continue to be read and reported, but no testing is done.
- (9) Options work in any order

- Input streams:
  - 0: Data 1
  - 1: Data 2

- Output streams:
  - 0: Result (single record, possibly multiple lines)
  - 1: Last primary record read at first no match, or end of stream
  - 2: Last secondary record read at first no match, or end of stream

**configure**

Set and Query CMS Pipelines Configuration Variables

- Not implemented in Netrexx Pipelines.
### console

**Read or Write the Terminal in Line Mode**

| +TERMinal+ | +EOF+-delimitedString+ | +PRfix+-delimitedString+ |
| NOEOF----------------+ | Prompt+ |

#### CMSonly:
- +----------------+- (2)- |
- DIRECT--------- |
- ASYNchronously+ |
- DARK---------- |

- (1) NetRexx only
  - On first stage, delimitedString is put out as a prompt
  - On other stages, each line is prefixed with delimitedString
  - Outout to next stage does NOT include delimitedString
  - Either keyword can be used for either stage
- (2) CMS only

### copy

**Copy Records, Allowing for a One Record Delay**

| +COPY---------+ |

### count

**Count Lines, Blank-delimited Words, and Bytes**

| +COUNT--+-+-+-CHARACTErS-+-+-+------ |
| +-CHARS------+ |
| +-BYTES------+ |
| +-WORDS--------+ |
| +-LINES--------+ |
| +-RECORDS-- |
| +-MINline--------+ |
| +-MAXline-------+ |

### cp

**Issue CP Commands, Write Response to Pipeline**

- Not implemented in Netrexx Pipelines.

### crc

**Compute Cyclic Redundancy Code**

- Not implemented in Netrexx Pipelines.

### dam

**Pass Records Once Primed**

| +DAM---------+ |

### dateconvert
dateconverndateconv
dateconv

**Convert Date Formats**

3.09

3.11

Read or Write the Terminal in Line Mode

(1) NetRexx only
On first stage, delimitedString is put out as a prompt
On other stages, each line is prefixed with delimitedString
Outout to next stage does NOT include delimitedString
Either keyword can be used for either stage

(2) CMS only

Copy Records, Allowing for a One Record Delay

Count Lines, Blank-delimited Words, and Bytes

Issue CP Commands, Write Response to Pipeline

Not implemented in Netrexx Pipelines.

Compute Cyclic Redundancy Code

Not implemented in Netrexx Pipelines.

Pass Records Once Primed

Convert Date Formats
(1): SPACE is optional here.
(2): Not implemented in NetRexx Pipelines at this time; mainly mainframe useful only.
(3): NetRexx Pipelines uses IRange which gives a superset of range options.
(4): NetRexx Pipelines only. What time to assume if blank time on input.
(5): NetRexx Pipelines only.
  ○ Use current local date time.
  ○ Any Inputrange is ignored.
  ○ Any output format can be used.
  ○ PREFACE Write the date record before passing the input to the output.
<table>
<thead>
<tr>
<th>deal</th>
<th>Pass Input Records to Output Streams Round Robin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Diagram showing the deal pipeline]</td>
</tr>
<tr>
<td></td>
<td>• (1) Not yet in NetRexx Pipelines</td>
</tr>
<tr>
<td></td>
<td>• (2) Not CMS</td>
</tr>
<tr>
<td></td>
<td>• Since Java dispatches the stage threads, DEAL may not see a sever immediately, as the severing thread can get multitasked. This can make options like 'ANYEOF' work in unexpected ways.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deblock</th>
<th>Deblock External Data Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Diagram showing the deblock pipeline]</td>
</tr>
<tr>
<td></td>
<td>• CMS has many more mainframe centric formats that NetRexx Pipelines does not process.</td>
</tr>
<tr>
<td></td>
<td>• (1) Not CMS Pipelines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>decode64 64decode 3.11</th>
<th>Decode Base-64 Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Diagram showing the decode64 pipeline]</td>
</tr>
<tr>
<td></td>
<td>• NOTE: CMS is only 64DECODE, and does not have the options; it does MIME.</td>
</tr>
<tr>
<td></td>
<td>• BASIC - Output is mapped to a set of characters lying in A-Za-z0-9+/. The encoder does not add any line feed in output, the decoder rejects any character other than A-Za-z0-9+/.</td>
</tr>
<tr>
<td></td>
<td>• URL - Output is mapped to set of characters lying in A-Za-z0-9+. Output is URL and filename safe.</td>
</tr>
<tr>
<td></td>
<td>• MIME - Output is mapped to MIME friendly format. Output is represented in lines of no more than 76 characters each, uses a carriage return '\v' followed by a linefeed '\n' as the line separator. No line separator is present to the end of the encoded output.</td>
</tr>
<tr>
<td></td>
<td>• 3.11: New to NetRexx. Add MIME, BASIC, &amp; URL options.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>delay</th>
<th>Suspend Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>devinfo</th>
<th>Write Device Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dfsort</th>
<th>Interface to DFSORT/CMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>diage4</th>
<th>Submit Diagnose E4 Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict hash</th>
<th>Read or Write a Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Pipes for NetRexx only.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>dicta</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>hasha</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>dictr</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>hashr</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>dictw</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>hashw</td>
<td>Read or Write a Dictionary</td>
</tr>
<tr>
<td>digest</td>
<td>Compute a Message Digest</td>
</tr>
<tr>
<td>disk</td>
<td>Read a File</td>
</tr>
<tr>
<td>diskr</td>
<td>Read a File</td>
</tr>
<tr>
<td>file</td>
<td>Read a File Backwards</td>
</tr>
<tr>
<td>filer</td>
<td>Read a File Backwards</td>
</tr>
<tr>
<td>diska</td>
<td>Append to or Create a File</td>
</tr>
<tr>
<td>filea</td>
<td>Append to or Create a File</td>
</tr>
<tr>
<td>diskid</td>
<td>Map CMS Reserved Minidisk</td>
</tr>
<tr>
<td>disk</td>
<td>Read a File</td>
</tr>
<tr>
<td>diskr</td>
<td>Read a File</td>
</tr>
<tr>
<td>file</td>
<td>Random Access a File</td>
</tr>
<tr>
<td>filerandom</td>
<td>Random Access a File</td>
</tr>
<tr>
<td>diskslow</td>
<td>Read, Create, or Append to a File</td>
</tr>
<tr>
<td>fileslow</td>
<td>Read, Create, or Append to a File</td>
</tr>
<tr>
<td>diskupdate</td>
<td>Replace Records in a File</td>
</tr>
<tr>
<td>fileupdate</td>
<td>Replace Records in a File</td>
</tr>
<tr>
<td>display</td>
<td>Output to Web Browser</td>
</tr>
</tbody>
</table>
DISPLAY works similar to and as a replacement for CONSOLE for output. But instead of going to the terminal window, it goes to a HTML file browser tab. This allows for HTML+CSS tags to control fonts, colors, and layout. This stage can be found in examples-pipes.

To work, these are required outside Pipelines and NetRexx:
- A working HTML browser program
- The operating system to associate the filetype "html" with the browser, so the Pipelines stage "COMMAND PipeDisp.html" does call the browser and display the file.
- The system have a Temp directory, known to Java.

The DISPLAY stage overwrites the named file, by default PipeDisp.html, in the system Temp directory, then calls the COMMAND stage to display it. The file is not erased automatically by this stage.

Each DISPLAY stage invocation opens a new browser tab, which remains open.

The AS option causes the data to be surrounded by html tags.
- The default TEXT or PRE puts on <pre> and </pre>. Most browsers use:
  - Fixed width font
  - Display all the white spaces: line feeds and multiple spaces
- HTML uses <html> and </html>. Most browsers use:
  - Variable width font
  - Consolidate strings of white space into a single space
  - All the HTML tags
- TABLE uses <table> and </table>
  - Expects the data records to begin with <tr><td> (or <tr><th>)
- NOTAG uses <pre> & </pre>, but first converts all & characters to the entity &amp; and < characters to &lt; so HTML tags are not processed.
- NONE uses no extra tags. Most browsers use:
  - HTML display

OPTIONs QSTRING is included in the opening tag for the AS option. This could be CLASS, STYLE, or other options.

TITLE QSTRING adds "title"="delimitedString"<title> to the beginning of the output. This should show as the title in the tab.
Note: This officially should go into a HEAD section; here it won't be there. Most modern browsers will honor it anyplace.
If it is not honored as a tag, QSTRING will be the top line of the display.

STYLE QSTRING adds <link rel="stylesheet" href="QSTRING"/> to the beginning of the output. This should include and use the named stylesheet. The name may have relative path names, or be an absolute file name. If there are spaces, enclose quotes.
Note: This officially should go into a HEAD section; here it won't be there. Most modern browsers will honor it anyplace.
If it is not honored as a tag, it will not show -- except in the NOTAG option. The file itself is copied from its stated location into the system Temp directory, overwriting any existing file. This file is not erased automatically by this stage.

QSTRING: It is optional to enclose the name in quotes, but quotes are required if the name includes spaces.

FILETYPE may be used to change the default "html". This permits use of other types that MAY be preprocessed if the system, external to Pipelines, is set up to recognize it, for example, "JSP" or "PHP". A "dot" is optional; only one will suffice.
Note: filetypes other than .html may be handled by the system by some program other than the browser.

QSTRING: It is optional to enclose the type in quotes.

FILENAME may be used to write and display another file. It may include a path designation, either absolute or relative.
Relative path is based on the working directory. If no path is specified in the name, the system Temp directory, as determined by Java, is used.

QSTRING: It is optional to enclose the name in quotes, but quotes are required if the name includes spaces.

Records from the primary input stream are also put out on the primary output stream unchanged, if it is connected.
**drop**

Discard Records from the Beginning or the End of the File

```plaintext
+--FIRST--+  ++-1---------+
      +--DROP------------------------+---
      ++-LAST--+  ++-number-(1)+  ++-BYTES--
      ++-**------+
```

- (1) CMS: must be positive.
  NetRexx Pipelines: negative reverses FIRST/LAST, so DROP FIRST -3 is the same as DROP LAST 3.

**duplicate**

Copy Records

```plaintext
+--**-------+
      +--DUPlicate-(1)++-+---------+---
      ++-number++
      ++-**------+
      ++- -1-----+
```

- (1) CMS is DUPlicat due to 8-character name limitation

**elastic**

Buffer Sufficient Records to Prevent Stall

```plaintext
+--ELASTIC---------+---
```

**encode64**

Encode to Base-64 Format

```plaintext
NetRexx

+--**---+
      +--64ENCODE++-+---------+---
      ++-ENCODE64++
      ++-BASIC++
      ++-URL---+
```

- NOTE: CMS is only 64DECODE, and does not have the options; it does MIME.
- BASIC - Output is mapped to a set of characters lying in A-Za-z0-9+/_. The encoder does not add any line feed in output, the decoder rejects any character other than A-Za-z0-9+/_.
- URL - Output is mapped to a set of characters lying in A-Za-z0-9+_. Output is URL and filename safe.
- MIME - Output is mapped to MIME friendly format. Output is represented in lines of no more than 76 characters each, uses a carriage return 'v' followed by a linefeed 'n' as the line separator. No line separator is present to the end of the encoded output.

**eofback**

Run an Output Device Driver and Propagate End-of-?le Backwards

- Not implemented in Netrexx Pipelines.

**escape**

Insert Escape Characters in the Record

- Not implemented in Netrexx Pipelines.

**fanin**

Concatenate Streams

```plaintext
+--FANIN--------+
      |
      +--stream++++
```

**faninany**

Copy Records from Whichever Input Stream Has One

```plaintext
+--FANINANY------
```

**fanintwo**

Pass Records to Primary Output Stream
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fanout</td>
<td>Copy Records from the Primary Input Stream to All Output Streams</td>
</tr>
<tr>
<td>fanouttwo</td>
<td>Copy Records from the Primary Input Stream to Both Output Streams</td>
</tr>
<tr>
<td>fbaread</td>
<td>Read Blocks from a Fixed Block Architecture Drive</td>
</tr>
<tr>
<td>fbawrite</td>
<td>Write Blocks to a Fixed Block Architecture Drive</td>
</tr>
<tr>
<td>fblock</td>
<td>Block Data, Spanning Input Records</td>
</tr>
<tr>
<td>file</td>
<td>Read a File</td>
</tr>
<tr>
<td>filer</td>
<td>Append to or Create a File</td>
</tr>
<tr>
<td>disk</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>diskr</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>filea</td>
<td>Read a CMS file backwards</td>
</tr>
<tr>
<td>diska</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>fileback</td>
<td>Read a CMS file backwards</td>
</tr>
<tr>
<td>diskback</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>filedirectory</td>
<td>Read or Write an OpenExtensions File that is Already Open</td>
</tr>
<tr>
<td>filerandom</td>
<td>Read specific records from a CMS file</td>
</tr>
<tr>
<td>diskrandom</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>fileslow</td>
<td>Read, Create, or Append to a File</td>
</tr>
<tr>
<td>diskslow</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>filetoken</td>
<td>Read or Write an SFS File That is Already Open</td>
</tr>
<tr>
<td>fileupdate</td>
<td>Change records in a CMS file</td>
</tr>
<tr>
<td>fileupdate</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>fillup</td>
<td>Pass Records To Output Streams</td>
</tr>
<tr>
<td>fillup</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>filterpack</td>
<td>Manage Filter Packages</td>
</tr>
<tr>
<td>find</td>
<td>Select Lines by XEDIT Find Logic</td>
</tr>
<tr>
<td>fitting</td>
<td>Source or Sink for Copipe Data</td>
</tr>
<tr>
<td>fmtfst</td>
<td>Format a File Status Table (FST) Entry</td>
</tr>
<tr>
<td>frlabel</td>
<td>Select Records from the First One with Leading String</td>
</tr>
<tr>
<td>fromlabel</td>
<td>Select Records from the First One with Leading String</td>
</tr>
<tr>
<td>frtarget</td>
<td>Select Records from the First One Selected by Argument Stage</td>
</tr>
<tr>
<td>fullscrq</td>
<td>Write 3270 Device Characteristics</td>
</tr>
<tr>
<td>fullscrs</td>
<td>Format 3270 Device Characteristics</td>
</tr>
<tr>
<td>gate</td>
<td>Pass Records Until Stopped</td>
</tr>
<tr>
<td>gather</td>
<td>Copy Records From Input Streams</td>
</tr>
<tr>
<td>getfiles</td>
<td>Read Files</td>
</tr>
<tr>
<td>getovers</td>
<td>Write the Contents of Objects</td>
</tr>
<tr>
<td>getstems</td>
<td>Write the Contents of Members of Stems</td>
</tr>
</tbody>
</table>

- **getovers**
  - Input stream 0 should contain rexx objects. The getovers stage will output the index and contents of the stem on stream 0. If output stream 1 is connected, the root is placed there. Any severed streams will cause the stage to exit. Passing a non-rexx object will cause the stage to exit with return code 13. Pipes for NetRexx only.

- **getstems**
  - Input stream 0 should contain rexx objects containing stems. The getstems stage will output the contents of the stems on stream 0. If output stream 1 is connected, the root is placed there. Any severed streams will cause the stage to exit. Passing a non-rexx stem object will cause the stage to exit with return code 13. Pipes for NetRexx only.
select lines by a regular expression

NetRexx Pipes for NetRexx only.
Records matching the RegEx are put out on primary output.
Records not matching are put out on secondary, if connected, or discarded.

1. Regex_string is a Java RegEx expression. Null string passes all records.
2. Records are prefaced with records number, 10 characters, right justified.
3. Number of records put out after a matching record.
4. Number of records put out before and after a matching record.
5. Inserted before a group of "before records" or the found record with "after records."
6. Send all matching records (no numbers) to tertiary output stream, if connected.
7. Only a count of matches is put out on the primary output stream. (Other options probably should not be used with this.)

hash dict
   Read or Write a Dictionary
   * Pipes for NetRexx only.

hasha dicta
   Read or Write a Dictionary

hashr dirct
   Read or Write a Dictionary

hashw dictw
   Read or Write a Dictionary

help ?
   Display Help for Pipelines
   * Not implemented in Netrexx Pipelines.

hfs
   Read or Append File in the Hierarchical File System
   * Not implemented in Netrexx Pipelines.

hfsdirectory
   Read Contents of a Directory in a Hierarchical File System
   * Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>hfsquery</td>
<td>Write Information Obtained from OpenExtensions into the Pipeline</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hfsreplace</td>
<td>Replace the Contents of a File in the Hierarchical File System</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hfsstate</td>
<td>Obtain Information about Files in the Hierarchical File System</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hfsexecute</td>
<td>Issue OpenExtensions Requests</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hlasm</td>
<td>Interface to High Level Assembler</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hlasmerr</td>
<td>Extract Assembler Error Messages from the SYSADATA File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>hole</td>
<td>Destroy Data</td>
<td></td>
</tr>
<tr>
<td>hostbyaddr</td>
<td>Resolve IP Address into Domain and Host Name</td>
<td></td>
</tr>
<tr>
<td><strong>3.09</strong></td>
<td>(1) Optional parameter not present in VM/CMS version</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCLUDEIP - Also include the IP address along with the hostname. Output: &lt;hostname&gt;/&lt;ip address&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: dns.google/8.8.8.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known issues: The underlying Java method getByName/getHostName does not appear to handle IPv6 addresses in a known and consistent manner. Could be related to a host configuration issue but googling shows odd and inconsistent results for getting around this.</td>
<td></td>
</tr>
<tr>
<td>hostbyname</td>
<td>Resolve a Domain Name into an IP Address</td>
<td></td>
</tr>
<tr>
<td><strong>3.09</strong></td>
<td>(1) Optional parameter not present in CMS Pipelines</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCLUDENAME - Also include the name of the host on output. Output: &lt;hostname&gt;/&lt;ip address&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: dns.google/8.8.8.8</td>
<td></td>
</tr>
<tr>
<td>hostid</td>
<td>Write TCP/IP Default IP Address</td>
<td></td>
</tr>
<tr>
<td><strong>3.09</strong></td>
<td>(1) The USERid option available under CMS Pipelines is not applicable and is ignored in NetRexx Pipelines</td>
<td></td>
</tr>
<tr>
<td>hostname</td>
<td>Write TCP/IP Host Name</td>
<td></td>
</tr>
<tr>
<td><strong>3.09</strong></td>
<td>(1) Optional parameter not present in VM/CMS version</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2) The USERid option available under CMS is not applicable and is ignored in NetRexx Pipelines</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arguments: INCLUDEIP - include the IP address of the system in the response in the form &lt;hostname&gt;/&lt;ip address&gt;</td>
<td></td>
</tr>
</tbody>
</table>
**htmlrows**  
Convert rows to HTML format

```plaintext
-NetRexx-

��-SEParator ","------+

▌-HTMLROWS-+------------------------+------------------------
    +-ROW-QString-+-SEParator-QString+
    +------------------------+

▌-------------------------+

▌-HEAD-QString-+-DATA-QString+

▌-------------------------+
```

- HTMLROWs reads rows from its primary input stream and writes them to its primary output stream, altering them to proper HTML tags for TABLE ROWS.
- I.e., it converts
  abc,mnop,xyz into
  `<tr><td>abc</td><td>mnop</td><td>xyz</td></tr>`
- The SEPARATOR QString, by default the comma character, can be specified.
- There are options to put additional data inside the tags. This could be used for class or style tag options, for example
  - ROW QString : puts its information into the `<tr>`-tags
  - DATA QString : puts its information into the `<td>`-tags
  - HEAD QString : puts its information into the `<th>`-tags (1)
- QString is a quoted string of characters. The quote marks may be either single or double, but must match. If there are spaces in the string, the quote marks are optional.
- (1) If there is a HEAD option, the first row read has `<th>`-tags instead of `<td>`-tags. It must have a QString of at least "". Succeeding rows have the standard `<td>`-tags.

**httpsplit**  
Split HTTP Data Stream

- Not implemented in Netrexx Pipelines.

**iebcopy**  
Process IEBCOPY Data Format

- Not implemented in Netrexx Pipelines.

**if**  
Process Records Conditionally

- Not implemented in Netrexx Pipelines.

**immcmd**  
Write the Argument String from Immediate Commands

- Not implemented in Netrexx Pipelines.

**insert**  
Insert String in Records

```plaintext
▌-BEFORE-+

▌-------------------------+

▌-AFTER-+-inputRange--+
```

- insert a string into a record before or after the record content. Will be much more efficient than specs especially if the Byte[]

**inside**  
Select Records between Labels

```plaintext
▌--------------------------
    +delimitedString--+-number--------+

▌-------------------------+
```

**instore**  
Load the File into a storage Buffer

- Not implemented in Netrexx Pipelines.

**ip2socka**  
Build sockaddr_in Structure

- Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| ispf    | Access ISPF Tables
                     - Not implemented in Netrexx Pipelines. |
| jeremy  | Write Pipeline Status to the Pipeline
                     - Not implemented in Netrexx Pipelines. |
| join    | Join Records |
| joincont| Join Continuation Lines |
| juxtapose| Preface Record with Marker |
| ldrtbls | Resolve a Name from the CMS Loader Tables
                     - Not implemented in Netrexx Pipelines. |
| listcat | Obtain Data Set Names
                     - Not implemented in Netrexx Pipelines. |
| listdsi | Obtain Information about Data Sets
                     - Not implemented in Netrexx Pipelines. |
| listispf| Read Directory of a Partitioned Data Set into the Pipeline
                     - Not implemented in Netrexx Pipelines. |
| listpds | Read Directory of a Partitioned Data Set into the Pipeline
                     - Not implemented in Netrexx Pipelines. |
| listzip | List the Files in a Zipped File |
| literal | Write the Argument String |

### ispf
Access ISPF Tables
- Not implemented in Netrexx Pipelines.

### jeremy
Write Pipeline Status to the Pipeline
- Not implemented in Netrexx Pipelines.

### join
Join Records

```
+----+--------+
|   | COUNT   |
+----+--------+
```

### joincont
Join Continuation Lines

```
+-----+-------------------+--+
|     | delimitedString   |
+-----+-------------------+--+
```

### juxtapose
Preface Record with Marker

```
+----+-------------------+--+
|    | COUNT             |
+----+-------------------+--+
```

### ldrtbls
Resolve a Name from the CMS Loader Tables
- Not implemented in Netrexx Pipelines.

### listcat
Obtain Data Set Names
- Not implemented in Netrexx Pipelines.

### listdsi
Obtain Information about Data Sets
- Not implemented in Netrexx Pipelines.

### listispf
Read Directory of a Partitioned Data Set into the Pipeline
- Not implemented in Netrexx Pipelines.

### listpds
Read Directory of a Partitioned Data Set into the Pipeline
- Not implemented in Netrexx Pipelines.

### listzip
List the Files in a Zipped File

```
NetRexx-
```

### literal
Write the Argument String

```
+----+--------+
|    | string |
+----+--------+
```
Select Lines that Contain a String

(1) Not in NetRexx Pipelines, yet.

Find Records in a Reference Using a Key Field

- in stream 0 are detail records
- in stream 1 are master records
- in stream 2 adds to masters
- in stream 3 delete from masters
- out stream 0 are matched records
- out stream 1 are unmatched detail records
- out stream 2 are unmatched or counted master records
- out stream 3 deleted masters
- out stream 4 duplicate masters
- out stream 5 unmatched master deletes
- lookup does not consider an unconnected output stream an error. It does propogate EOFs from output streams.
Find Records in a Reference Using a Key Field

- in stream 0 are detail records
- in stream 1 are master records
- in stream 2 adds to masters
- in stream 3 delete from masters
- out stream 0 are matched records
- out stream 1 are unmatched detail records
- out stream 2 are unmatched or counted master records
- out stream 3 deleted masters
- out stream 4 duplicate masters
- out stream 5 unmatched master deletes
- lookup does not consider an unconnected output stream an error. It does propagate EOFs from output streams.

Generate a Macro Library from Stacked Members in a COPY File

- Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapmdisk</td>
<td>Map Minidisks Into Data spaces</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mctoasa</td>
<td>Convert CCW Operation Codes to ASA Carriage Control</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mdiskblk</td>
<td>Read or Write Minidisk Blocks</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mskrandom</td>
<td>Random Access a CMS File on a Mode</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mskslow</td>
<td>Read, Append to, or Create a CMS File on a Mode</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mskupdate</td>
<td>Replace Records in a File on a Mode</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>members</td>
<td>Extract Members from a Partitioned Data Set</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>merge</td>
<td>Merge Streams</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>mqsc</td>
<td>Issue Commands to a WebSphere MQ Queue Manager</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>nfind</td>
<td>Select Lines by XEDIT NFind Logic</td>
<td></td>
</tr>
<tr>
<td>notfind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ninside</td>
<td>Select Records Not between Labels</td>
<td></td>
</tr>
<tr>
<td>notinside</td>
<td>3.09</td>
<td></td>
</tr>
<tr>
<td>nlocate</td>
<td>Select Lines That Do Not Contain a String</td>
<td></td>
</tr>
<tr>
<td>notlocate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noEofBack</td>
<td>Pass Records and Ignore End-of-file on Output</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>nop</strong></td>
<td>No Operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pipes for NetRexx only.</td>
<td></td>
</tr>
<tr>
<td><strong>not</strong></td>
<td>Run Stage with Output Streams Inverted</td>
<td></td>
</tr>
<tr>
<td><strong>nfind</strong></td>
<td>Select Lines by XEDIT NFind Logic</td>
<td></td>
</tr>
<tr>
<td><strong>ninside</strong></td>
<td>Select Records Not between Labels</td>
<td></td>
</tr>
<tr>
<td><strong>nlocate</strong></td>
<td>Select Lines that Do Not Contain a String</td>
<td></td>
</tr>
<tr>
<td><strong>ncext</strong></td>
<td>Call a Nucleus Extension</td>
<td></td>
</tr>
<tr>
<td><strong>optcdj</strong></td>
<td>Generate Table Reference Character (TRC)</td>
<td></td>
</tr>
<tr>
<td><strong>outside</strong></td>
<td>Select Records Not between Labels</td>
<td></td>
</tr>
<tr>
<td><strong>outstore</strong></td>
<td>Unload a File from a storage Buffer</td>
<td></td>
</tr>
<tr>
<td><strong>over</strong></td>
<td>Write the Values of Stems</td>
<td></td>
</tr>
</tbody>
</table>

- **nop**:
  - No Operation
  - Pipes for NetRexx only.

- **not**:
  - Run Stage with Output Streams Inverted

- **nfind**:
  - Select Lines by XEDIT NFind Logic

- **ninside**:
  - Select Records Not between Labels

- **nlocate**:
  - Select Lines that Do Not Contain a String

- **ncext**:
  - Call a Nucleus Extension

- **optcdj**:
  - Generate Table Reference Character (TRC)

- **outside**:
  - Select Records Not between Labels

- **outstore**:
  - Unload a File from a storage Buffer

- **over**:
  - Write the Values of Stems
    - Obsolete. Now use varover. over is now an alias for overlay.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>overlay</strong></td>
<td>Overlay Data from Input Streams</td>
</tr>
<tr>
<td><strong>HOLD</strong></td>
<td>keeps the last record from each stream, except primary, and uses it if the stream ends.</td>
</tr>
<tr>
<td><strong>TRANSPARENT</strong></td>
<td>means that character can be different from the PAD character. If omitted, it is the same as PAD character.</td>
</tr>
<tr>
<td><strong>dstream</strong></td>
<td>can be used instead of a non-primary stream.</td>
</tr>
<tr>
<td><strong>(1)</strong></td>
<td>NetRexx Pipelines only</td>
</tr>
<tr>
<td><strong>(2)</strong></td>
<td>same as highest (+1) stream; implies HOLD</td>
</tr>
<tr>
<td><strong>overstr</strong></td>
<td>Process Overstruck Lines</td>
</tr>
<tr>
<td><strong>pack</strong></td>
<td>Pack Records as Done by XEDIT and COPYFILE</td>
</tr>
<tr>
<td><strong>pad</strong></td>
<td>Expand Short Records</td>
</tr>
<tr>
<td><strong>parcel</strong></td>
<td>Parcel Input Stream Into Records</td>
</tr>
<tr>
<td><strong>parse 3.09</strong></td>
<td>Rearrange Contents of Records</td>
</tr>
<tr>
<td><strong>pause</strong></td>
<td>Signal a Pause Event</td>
</tr>
<tr>
<td><strong>pdsdirect</strong></td>
<td>Write Directory Information from a CMS Simulated Partitioned Data Set</td>
</tr>
</tbody>
</table>
Select Lines that Satisfy a Relation

- **NetRexx**

```
+-NOPAD----+  

▶+-PICK+-+---------------+-+  

+-PAD-xorc+  +-ANYcase-(1)+  

+-CASEANY----+  

+-CASEIGNORE-+  

+-CASELESS----+  

+-IGNORECASE-+  

▶+-+---------------+-+---------
```

- **inputRange**

```
+-^==-+  +-=---+  +-=---+  

+-\==--+(2)  

+-/=--+(2)  

+-/=--+(2)  

+-/=--+(2)  
```

- **delimitedString**

```
+-<=--+
```

- **inputRange**

```
+-><=-+
```

- **<==--+**

- (1) Can be before PAD/NOPAD. Deprecated.
- (2) The backslash (\) may need to be escaped, doubled, in some systems shells.
Select Lines that Satisfy a Relation

CMS

+-NOPAD------+

-+PAD--korc+- +ANYcase----+
      +CASEANY-----
      +CASEIGNORE--
      +CASELESS----
      +IGNORECASE--

-+-FROM+-+-+-+-+
   |   +-TO----  +AFTER++ |
   |   +WHILE----------|
   +-| Fromto |----------------------------------|

List:
|+-FROM--------| List |+-TO--------| List |
   |  | +AFTER++  |
   |  | +AFTER++  |
   |  | +COUNT--number--------|

Test:
|+-| RangeString |+-| NonEqualOp |+-| RangeString |
   |  | +EqualOp |+-| CommaList |

CommaList:
   -=-,------------------
   |----| RangeString |

RangeString:
|+-| inputRange-------|
   |  | +delimitedString++
   |  | +number+--------+

Character Operators:
== ^== \== /== <<< <= >>>= IN NOTIN

Numeric Operators:
= ^= < <= >>=
Select Lines that Satisfy Relations using Rexx Parse

-NetRexx-

-ONE-+---(2)-+---parse_Dstring-+---logic_Dstring-+---ELSE-(1)-+

-ALL-+---SINGLE-+

-Records are parsed via the parse_delimited_string.
-Variables are named $n, where n is 1 to 9.
-The values of the variables are put into the logic_delimited_string replacing $n and evaluated. If TRUE, the record is put on the stream numbered by the dstring's position.
-The stream for a Dstring of ELSE is used if no previous logic Dstring is TRUE.
-If there is no specific ELSE, there is an implied one at the end; if that stream is not connected, the record is discarded.
-If ONE then the record is put out on, at most, one stream: the first one matched.
-If ALL then the record is put out on all streams matched.
-If SINGLE then the records are all put out on the primary output stream.
-The parse_delimited_string and logic_delimited_string(s) follow normal NetRexx rules.
-(1) Implied ELSE after last specified dstring.
-(2) Up to 10 logic_Dstrings may be specified to go to up to 11 output streams (including an implied ELSE).
-Not implemented in CMS Pipelines.

Pickparse permits selecting records by a NetRexx logical expression, using parts of the record selected by a Rexx PAR...template.

A simple example has two delimited strings, a Rexx template and a logical expression:

pickparse / . $3 . 50 $5 +5 / / $3 < $5 /

The parse template selects the 3rd word, and the 5 characters starting in column 50. The variable names are a dollar sign and a digit. Then those variables can be used in the logic expression. When run, and records matching the logic expression are written to the primary output stream, others to the secondary. If either stream is not connected, the corresponding records are discarded.

There can be multiple logic expressions, each in its own delimited string. Parenthetical expressions may be used. Records are matched to each in turn. Any records matching are written to that output stream, if connected.

With the option ONE, the default, each record is written to one output stream: the first one it matches. With the option ALL matching goes on and a record could be written to multiple output streams.

There is an implicit or explicit ELSE as the last logic expression. Records that have not matched any of the previous expressions match this and are written or discarded depending on if the stream is connected or not.

The parse template can define up to 9 separate zones, $1 to $9. The variables $_n are also available for the logic expressions; they are the values from the previous record. Initially these are "".

There can be up to 10 output streams defined, and up to 9 logic expressions plus ELSE.

---

**pipcmd**  
**Issue Pipeline Commands**

- Not implemented in Netrexx Pipelines.

**pipestop**  
**Terminate Stages Waiting for an External Event**

- Not implemented in Netrexx Pipelines.

**polish**  
**Reverse Polish Expression Parser**

- Not implemented in Netrexx Pipelines.

**predselect**  
**Control Destructive Test of Records**

- Not implemented in Netrexx Pipelines.

**preface**  
**Put Output from a Device Driver before Data on the Primary Input Stream**

- Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>prefix</th>
<th>Stop and Run a Stage First, Before Continuing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-NetRexx-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>--PREFIX--</strong> <strong>string</strong>-<strong>---</strong>&gt;**</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Blocks its primary input and excutes stage supplied as an argument. The output from this stage are put to the primary input stream. When its compete the primary input is shorted.</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in CMS Pipelines.</td>
</tr>
<tr>
<td>printmc</td>
<td>Print Lines</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>punch</td>
<td>Punch Cards</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>qpdecode</td>
<td>Decode to Quoted-printable Format</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>qpencode</td>
<td>Encode to Quoted-printable Format</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>qsam</td>
<td>Read or Write Physical Sequential Data Set through a DCB</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>query</td>
<td>Obtain Information From Pipelines</td>
</tr>
<tr>
<td></td>
<td><strong>+-VERSION------+</strong></td>
</tr>
<tr>
<td></td>
<td><strong>---Query---</strong> <strong>-----------</strong></td>
</tr>
<tr>
<td></td>
<td><strong>+-LEVEL--------</strong></td>
</tr>
<tr>
<td></td>
<td><strong>+-SOURCE- (1)---</strong></td>
</tr>
<tr>
<td></td>
<td><strong>+-MSGLEVEL- (2) --</strong></td>
</tr>
<tr>
<td></td>
<td><strong>+-MSGLIST- (2) --</strong></td>
</tr>
<tr>
<td></td>
<td>- (1) Not CMS</td>
</tr>
<tr>
<td></td>
<td>- (2) Not NetRexx Pipelines</td>
</tr>
<tr>
<td>random</td>
<td>Generate Pseudorandom Numbers</td>
</tr>
<tr>
<td>3.09</td>
<td><strong>---RANDOM---</strong> <strong>---------------------</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>+-max_number-</strong> <strong>---------------------</strong></td>
</tr>
<tr>
<td></td>
<td><strong>+-seed_snumber-</strong></td>
</tr>
<tr>
<td></td>
<td>- NetRexx Pipelines will be a different sequence than CMS gives with the same seed.</td>
</tr>
<tr>
<td>reader</td>
<td>Read from a Virtual Card Reader</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>readpds</td>
<td>Read Members from a Partitioned Data Set</td>
</tr>
<tr>
<td></td>
<td>- Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>
### Select Lines by a Regular Expression

NetRexx Pipelines only.
Records matching the RegEx are put out on primary output.
Records not matching are put out on secondary, if connected, or discarded.

1. Regex string is a Java RegEx expression. Null string passes all records.
2. Records are prefaced with records number, 10 characters, right justified.
3. Number of records put out after a matching record.
4. Number of records put out before and after a matching record.
5. Inserted before a group of "before records" or the found record with "after records."
6. Send all matching records (no numbers) to tertiary output stream, if connected.
7. Only a count of matches is put out on the primary output stream. (Other options probably should not be used with this.)

### regen

Replace Runs of Blanks with Tabulate Characters
- Not implemented in Netrexx Pipelines.

### reverse

Reverse Contents of Records

### rexx

Run a REXX Program to Process Data
- Not implemented in Netrexx Pipelines.

### rexxvars

Retrieve Variables from a REXX or CLIST Variable Pool
- Not implemented in Netrexx Pipelines.

### runpipe

Issue Pipelines, Intercepting Messages
- Not implemented in Netrexx Pipelines.

### scm

Align REXX Comments
- Not implemented in Netrexx Pipelines.

### sec2greg

Convert Seconds Since Epoch to Gregorian Timestamp
- Not implemented in Netrexx Pipelines.

### serialize

Convert Objects to a Single Text String
- NetRexx

- (class) if class is specified deserialize input to objects of this type otherwise serialize input objects.
- Pipes for NetRexx only.
- For some reason readObject does not like more than one object network in its stream. Block multiple objects. See examples/sertest.njp
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Implementation Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>sfsback</td>
<td>Read an SFS File Backwards</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>sfsdirectory</td>
<td>List Files in an SFS Directory</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>sfsrandom</td>
<td>Random Access an SFS File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>sfsupdate</td>
<td>Replace Records in an SFS File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

**snake 3.09**
Build Multicolumn Page Layout

```
>>>--SNAKE--number_cols--+-number_rows--+-page separator_DString--(1)--
```

- (1) NetRexx Pipelines only. Appears first, last, and between pages.
- Avoid `\` as escape terms maybe added in the future. `\n` for newline is OK.
- Your system may require `\n`.

**socka2ip**
Format sockaddr_in Structure

- Not implemented in Netrexx Pipelines.

**sort**
Order Records

```
>>>--SORT--+-REXX------+-10000+--inputRange+-
+(+-+---------------+-+-)-+
+class-(2)+ +size--+

+Ascending-(1)+

+Descending-(1)+ +SINGLEOK-(3)+
```

- (1) May come before inputRange, for backwards compatibility.
- (2) Requires that you implement another sortClass with a name beginning with 'sort'
- (3) Suppresses error message if only one record to sort for Rexx objects.
- Uses sortClass class as Interface Class for Generic Sort Objects and sortRexx class to Sort Rexx Text Objects

---

**CMS**
Order Records

```
>>>--SORT--+-----+-nopad--+-

+COUNT--+-PAD-xorc+-ANYcase+-
+UNIQue+-

+Ascending+-+

+Descending+-
```

- `+-inputRange+-`
(0) The order is the reverse of CHANGE!
(1) the replacement char/string
(2) the char/chars that will be stripped and replaced
(3) NetRexx Pipelines only, not CMS. The dstring is treated as a single unit for stripping or replacing
Blanks are optional in this position.

Blanks are not allowed here.

CMS only. Not yet implemented in NetRexx Pipelines

NetRexx Pipelines only. Not yet implemented in CMS

READ is giving the same output as READSTOP when the streams are different length.

This senses if it is the first stage, but comment stages will fool it into not producing any output.

---+-------+--+--------------------+------|
| STRIP-+ | B2C----------------+ |
| -B2D-+ | B2X----------------+ (4) |
| +C2B+ | +C2D----------------+ |
| +C2F-+ | C2I----------------+ (3) |
| +C2P-+ | +C2P----------------+ (3) |
| (2) | (scale)-- | (3) |
| ----+ | B2V----------------+ (3) |
| +C2X-+ | +C2X----------------+ |
| +D2C-+ | +D2C----------------+ |
| +F2C-+ | F2C----------------+ (3) |
| +I2C-+ | +I2C----------------+ |
| +P2C-+ | +P2C----------------+ |
| (2) | (scale)-- | (3) |
| ----+ | V2C----------------+ (3) |
| +X2B-+ | +X2B----------------+ |
| +X2C-+ | +X2C----------------+ |
| +X2D-+ | +X2D----------------+ |
| +f2t-+ | +f2t----------------+ |
| -LOWER-+ | -LOWER-------------+ (4) |
| +UPPER-+ | +UPPER-------------+ (4) |
| +STRING-+ | +STRING-------------+ (4) |

---+-Next-+-------+-------+--|
|      |      |      |      | (2) |
| .n   |      |      |      | |
| NEXTWord-+-+-------+-+ |
| NWord----+ | | |
| columnrange---------+ |

---+--------+--|
| Left---+ |
| Center-+ |
| Centre-+ |
| Right--+ |

Ranges (cnumberrange, fnumberrange (3), wnumberrange):

---+---snumber-+- (2) +number-+-|
|      |      |      |      | (2) |
| ---*-----+ | number-+-|
|      |      |      |      | (2) |
| ---*-----+ | snumber-+-|
|      |      |      |      | (2) |

---+-------+--+--------------------+------|
| spill | Spill Long Lines at Word Boundaries |

---|---|
| Not implemented in Netrexx Pipelines. |
Split Records Relative to a Target

- ANYcase----
- MINimum--number--
- CASEANY-----
- CASEIGNORE--
- IGNORECASE---
- CASELESS----

- AT---------------------
- BLANK-----------------

BEFORE--
NOT--
target

- snumber--
- AFTER---

number--

target:
|---xrange------------------------|
|---STRing-----delimitedString--|

--ANYof--
Interface to SQL

uses jdbc to select from any jdbc enabled dbms
properties file (sqlselect.properties default) is read from the secondary input stream to find jdbcdriver name, url, use sample properties file:

#JDBC driver name
#Tue Feb 03 23:29:43 GMT+01:00 1998
jdbcdriver=com.imaginary.sql.msql.MsqlDriver
url=jdbc:msql://localhost:1114/TESTDB

# the following are not needed for some DBMS, ex: SQLite
user=db_user_name
pass=password_for_db

if this file is not found default (compiled in) values are used
(1) when using a sql select * (all columns) from the command line, quote the query as in
java pipes.compiler (query) "sql select * from dept | console"
(2) the netrexx/jdbc combination is extremely case sensitive for column and table names
(3) this sql_select_string executed, then statements are read from the primary input stream.
this is optional in NetRexx Pipelines only.
(4) CMS does not use the stream input
(5) NetRexx Pipelines only
(6) CMS Pipelines is implied HEADERS only.
(7) A Qword is an optionally quoted word. If it contains spaces, it must be quoted.
(8) EXPERIMENTAL Subject to change. DBMS is the kind of database, e.g. SQLite. DB_NAME is the file name. The used in place of URL and JDBCDRIVER. SQLite is the only one tested as of 8/15/20.
(9) the SQLSELECT stage uses HEADERS as the default.
(10) USER & PASS are needed for some DBMSs and not others, ex: SQLite.
(11) the count or other output from non-select statements goes to the secondary output stream if connected, or is discarded otherwise it goes to the primary.

Priority order for URL, JDBCDRIVER and DBMS, DB_NAME (first one found rules):
1. option in the SQL command string
2. from secondary input stream
3. from "sql.properties" file or from file specified by PROPERTIES option
4. Built-in

sqlcodes Write the last 11 SQL Codes Received
• Not implemented in Netrexx Pipelines.
### sqlselect

<table>
<thead>
<tr>
<th>Query</th>
<th>Database and Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) when using <code>sqlselect * (all columns)</code> from the command line, quote the query as in <code>java pipes.compiler (query)</code> from <code>dept</code></td>
<td>as in `java pipes.compiler (query)&quot;sqlselect * from dept</td>
<td>console&quot;`</td>
</tr>
<tr>
<td>(3) if no <code>sql_select_string</code> is specified, it is read from the primary input stream. This is optional in NetRexx Pipelines only. CMS does not use the stream input.</td>
<td></td>
<td>a maximum of only one record is ever read from the primary input stream.</td>
</tr>
<tr>
<td>(5)</td>
<td>Has Pipelines is implied HEADERS only.</td>
<td>NetRexx Pipelines only</td>
</tr>
<tr>
<td>(6) CMS Pipelines is implied HEADERS only.</td>
<td>(7)</td>
<td>A Qword is optionally quoted word. If it contains spaces, it must be quoted.</td>
</tr>
<tr>
<td>(8) EXPERIMENTAL Subject to change. DBMS is the kind of database, e.g. SQLite. DB_name is the file name. The used in place of URL and JDBC_DRIVER. SQLite is the only one tested as of 8/15/20.</td>
<td></td>
<td>(9) the SQL stage uses NOHEADERS as the default.</td>
</tr>
</tbody>
</table>
| (10) USER & PASS are needed for some DBMSs and not others, ex. SQLite. | | Priority order for URL, JDBC_DRIVER, DBMS, DB_NAME, USER, & PASS (first found rules):
1. option in the SQL command string
2. from secondary input stream
3. from "sqlselect.properties" file or from file specified by PROPERTIES option
4. Built-in |

### stack

- Read or Write the Program Stack
- Not implemented in NetRexx Pipelines.

### starmon

- Write Records from the "MONITOR System Service"
- Not implemented in NetRexx Pipelines.

### starsys

- Write Lines from a Two-way CP System Service
- Not implemented in NetRexx Pipelines.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| **state** | Provide Information about CMS Files  
- Not implemented in Netrexx Pipelines. |
| **state** | Verify that Data Set Exists  
- Not implemented in Netrexx Pipelines. |
| **statew** | Provide Information about Writable CMS Files  
- Not implemented in Netrexx Pipelines. |
| **stem** | Retrieve or Set Variables in a REXX or CLIST Variable Pool  
- NetRexx:  
  ▶▶--STEM--stem------------▶ ◄ |
| **stem** | Retrieve or Set Variables in a REXX or CLIST Variable Pool  
- CMS:  
  ▶▶--STEM--stem------------▶ ◄ |
| **stfile** | Store Facilities List  
- Not implemented in Netrexx Pipelines. |
| **storage** | Read or Write Virtual Machine Storage  
- Not implemented in Netrexx Pipelines. |
| **strasmfind** | Select Statements from an Assembler File as XEDIT Find  
- Not implemented in Netrexx Pipelines. |
| **strasmnfind** | Select Statements from an Assembler File as XEDIT NFind  
- Not implemented in Netrexx Pipelines. |
| **strfind** | Select Lines by XEDIT Find Logic  
- CMS:  
  ▶▶--STRFIND------------delimitedString----------▶ ◄ |
| **strfrlabel** | Select Records from the First One with Leading String  
- Not implemented in Netrexx Pipelines. |
### Select Records from the First One with Leading String

```plaintext
+-STRFROMLABEL--+  +-INCLUSIVE-+
               >>+-STRFROMLAbel-------------+--delimitedString-+-
               +-ANYcase----  +-EXCLUSIVE-+
               +-CASEANY----
               +-IGNORECASE-+
               +-CASEIGNORE-+
               +-CASELESS----+
```

### Remove Leading or Trailing Characters

```plaintext
+-STRIP-+------------------------+-
               +-LEADING--+-TO---+
               +-TRAILING--+-NOT-+
               +-BLANK----------------+
               +-CASE---+
               +-CASEANY----
               +-CASEIGNORE-+
               +-CASELESS----+
```

- (1) Not implemented in Netrexx Pipelines.

### Write the Argument String

```plaintext
+-STRITERAL-+---------------------------------+-
               +-PREFACE-+
               +-APPEND--+
               +-CONDITIONal-+ |+-IFEMPTY---+
```

### Select Lines by XEDIT NFInd Logic

```plaintext
+-STRNFIND--------+-delimitedString--------+-
               +-ANYcase----+
               +-CASEANY----
               +-IGNORECASE-+
               +-CASEIGNORE-+
               +-CASELESS----+
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>strtolabel</strong>&lt;br&gt;strtolabe&lt;br&gt;strtolab</td>
<td>Select Records to the First One with Leading String</td>
</tr>
<tr>
<td><strong>structure</strong></td>
<td>Manage Structure Definitions</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>strwhilelable</strong>&lt;br&gt;strwhilelabl&lt;br&gt;strwhilelab&lt;br&gt;strwhilela&lt;br&gt;strwhilele&lt;br&gt;strwhile &lt;br&gt;3.09</td>
<td>Select Run of Records with Leading String</td>
</tr>
<tr>
<td><strong>stsi</strong></td>
<td>Store System Information</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>subcom</strong></td>
<td>Issue Commands to a Subcommand Environment</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>substring</strong></td>
<td>Write substring of record</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>synchronise</strong>&lt;br&gt;synchronize</td>
<td>Synchronise Records on Multiple Streams</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>synchronise</strong>&lt;br&gt;synchronise</td>
<td>Synchronise Records on Multiple Streams</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>sysdsn</strong></td>
<td>Test whether Data Set Exists</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>sysout</strong></td>
<td>Write System Output Data Set</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>sysvar</strong></td>
<td>Write System Variables to the Pipeline</td>
</tr>
<tr>
<td></td>
<td>• Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td><strong>tag</strong>&lt;br&gt;3.11</td>
<td>Surrounds Input Records with a HTML tag and its End Tag</td>
</tr>
<tr>
<td></td>
<td>• Outputs a record: <code>&lt;tag string&gt;</code>, then passes through all records on its primary input, and finally a record: <code>&lt;/tag&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>• String may be empty.</td>
</tr>
</tbody>
</table>
### tags

**3.11**

Surrounds Input Records with HTML tags and their End Tags

```plaintext
TAGS----delimitedString--+-+----------------------+---
|                   +--+------------+--+----------+--------------
|                   +---delimitedString+++|
```

- Outputs a record for each delimitedString: `<delimitedString>`, then passes through all records on its primary input, a record for each, in reverse order: `</first_word_of_delimitedString>`.
- Any delimitedString may be a single word.

### take

Select Records from the Beginning or End of the File

```plaintext
FIRST+++-1----------+
TAKE----------------+
LAST+++number-----+-BYTES(1)+
                  +-snumber(2)+
                  +-*-----------+
```

- (1) CMS must be BYTES
- (2) Not CMS; NetRexx Pipelines: minus reverses first/last

### tape

Read or Write Tapes

- Not implemented in Netrexx Pipelines.

### tcphsum

Compute One's complement Checksum of a Message

- Not implemented in Netrexx Pipelines.

### tcpclient

Connect to a TCP/IP Server and Exchange Data

- Simple tcpclient implementation. The options implemented are similar to the CMS definition.
  - linger - wait a bit before terminating the last read (units SECONDS)
  - timeout - wait this long before timing reads out (units MS)
  - deblock - If deblock is omitted a copy stage is used.
  - group - similar to CMS. A delimited string containing a stage is expected. You can use a run of stages, but its is dangerous since you to know the stage sep character being used...
  - greeting - expect a greeting message and discard it
  - nodelay - use the nodelay option
  - keepalive - enable keep alive socket option
  - oneresponse - synchronize cmds/replys

### tcpdata

Read from and Write to a TCP/IP Socket

- Simple tcpdata implementation.
  - linger - wait a bit before terminating the last read (units SECONDS)
  - timeout - wait this long before timing reads out (units MS)
  - deblock - If deblock is omitted a copy stage is used.
  - group - similar to cmsg. A delimited string containing a stage is expected. You can use a run of stages, but its is dangerous since you need to know the stage sep character being used...
  - nodelay - use the nodelay option
  - oneresponse - synchronize requests/replies

### tcplistener

Listen on a TCP Port

- Simple tcplisten implementation. You can only supply the port and a timeout value, which is ignored unless tcplisten stream has been severed, in which case tcplisten terminates.
- If input stream 0 is connected, tcplisten does a peekto before calling the accept method. The object is consumed af output of the socket object returns.
Read or Write the Terminal in Line Mode

- (1) NetRexx only
  On first stage, delimitedString is put out as a prompt
  On other stages, each line is prefixed with delimitedString
  Outout to next stage does NOT include delimitedString
  Either keyword can be used for either stage
- (2) CMS only

threeway
Split record three ways

- Not implemented in Netrexx Pipelines.

timestamp
Prefix the Date and Time to Records

- (1) In CMS Pipelines, the delimited string is required. In NetRexx Pipelines, it defaults to // if no second string.

tokenise
tokenize
Tokenise Records

- (1) In CMS Pipelines, the first delimited string is required. In NetRexx Pipelines, it defaults to // if no second string.

tolabel
tolabe
tolab
Select Records to the First One with Leading String

trackblock
Build Track Record

- Not implemented in Netrexx Pipelines.

trackdeblock
Deblock Track

- Not implemented in Netrexx Pipelines.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Implementation Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>trackread</td>
<td>Read Full Tracks from ECKD Device</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>tracksquish</td>
<td>Squish Tracks</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>trackverify</td>
<td>Verify Track Format</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>trackwrite</td>
<td>Write Full Tracks to ECKD Device</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>trackxpand</td>
<td>Unsquish Tracks</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>translate</td>
<td>Transliterate Contents of Records</td>
<td>Not yet in Pipes for NetRexx</td>
</tr>
<tr>
<td>trfread</td>
<td>Read a Trace File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>truncate</td>
<td>Truncate the Record</td>
<td></td>
</tr>
<tr>
<td>truncat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>truncca</td>
<td></td>
<td></td>
</tr>
<tr>
<td>truncchop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>tso</td>
<td>Issue TSO Commands, Write Response to Pipeline</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>udp</td>
<td>Read and Write an UDP Port</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>unique</td>
<td>Discard or Retain Duplicate Lines</td>
<td></td>
</tr>
<tr>
<td>uniqu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unpack</td>
<td>Unpack a Packed File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>untab</td>
<td>Replace Tabulate Characters with Blanks</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>update</td>
<td>Apply an Update File</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>uro</td>
<td>Write Unit Record Output</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>utf</td>
<td>Convert between UTF-8, UTF-16, and UTF-32</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>var</td>
<td>Retrieve or Set a Variable in a REXX or CLIST Variable Pool</td>
<td>Pipes for NetRexx: this can only read vars</td>
</tr>
<tr>
<td>vardrop</td>
<td>Drop Variables in a REXX Variable Pool</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>varfetch</td>
<td>Fetch Variables in a REXX or CLIST Variable Pool</td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
| **varload** | Set Variables in a REXX or CLIST Variable Pool  
- Not implemented in Netrexx Pipelines. |
| **varover** | Write the Values of Stems  
- NetRexx Pipelines only; not CMS Pipelines |
| **varset** | Set Variables in a REXX or CLIST Variable Pool  
- Not implemented in Netrexx Pipelines. |
| **vchar** | Recode Characters to Different Length  
- Not implemented in Netrexx Pipelines. |
| **vector** | Read or Write an Array of Vectors  
- Pipes for NetRexx only. |
| **vectora** | Add to an Array of Vectors  
- Pipes for NetRexx only. |
| **vectorr** | Read From an Array of Vectors  
- Pipes for NetRexx only. |
| **vectorw** | Write to an Array of Vectors  
- Pipes for NetRexx only. |
| **verify** | Verify that Record Contains only Specified Characters  
- (1) NetRexx Pipelines only  
- (1) Examples: A-Z 0-9 c-g a4-ba; 16-bit Unicode characters or hex numbers  
- (1) Any number greater than zero, any order, of delimitedStrings and character-ranges are allowed. |
| **vmc** | Write VMCF Reply  
- Not implemented in Netrexx Pipelines. |
| **vmcdata** | Receive, Reply, or Reject a Send or Send/receive Request  
- Not implemented in Netrexx Pipelines. |
| **vmclient** | Send VMCF Requests  
- Not implemented in Netrexx Pipelines. |
| **vmclisten** | Listen for VMCF Requests  
- Not implemented in Netrexx Pipelines. |
| **waitdev** | Wait for an Interrupt from a Device  
- Not implemented in Netrexx Pipelines. |
<table>
<thead>
<tr>
<th>warp</th>
<th>Pipeline Wormhole</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>warplist</th>
<th>List Wormholes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>whilelabel 3.09</th>
<th>Select Run of Records with Leading String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="WHILELABEL diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wildcard</th>
<th>Select Records Matching a Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>writepds</th>
<th>Store Members into a Partitioned Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xab</th>
<th>Read or Write External Attribute Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xedit</th>
<th>Read or Write a File in the XEDIT Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xlate</th>
<th>Transliterate Contents of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="TRANSLATE diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xmsg</th>
<th>Issue XEDIT Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xpndhi</th>
<th>Expand Highlighting to Space between Words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not implemented in Netrexx Pipelines.</td>
</tr>
</tbody>
</table>
Write a Range of Characters

NetRexx uses UTF-16 (ASCII) and CMS uses EBCDIC

Run Selection Stage on Subset of Input Record
Index

Rexx, 12, 25, 27
arg, 26
binary, 29
catch, 7, 12, 26, 27, 29
class, 7, 12, 25, 26, 29
do, 7, 12, 26
do, 7, 12, 25, 27, 29
do, 7, 12, 25, 26, 29
do, 7, 12, 26, 29
do, 7, 25, 29
do, 7, 12, 26, 29
do, 7, 12, 25, 29
do, 7, 12, 29
do, 7, 26, 29
do, 7, 12, 26, 29
do, 7, 12, 25, 26, 29
do, 7, 12, 29
do
end, 7, 12, 25, 27, 29
end, 7, 12, 26, 29
end, 7, 12, 12, 26, 29
end, 7, 12, 26, 29
extends, 7, 12, 26, 29
final, 7, 26
forever, 7, 12, 26, 27, 29
import, 7, 12, 26, 29
import, 7, 12, 25, 27, 29
import, 7, 12, 25, 26, 29
import, 7, 12, 29
import, 7, 26, 29
import, 7, 29
import
loop, 7, 12, 25, 27, 29
loop, 7, 12, 25, 26, 29
loop, 7, 12, 29
loop, 7, 26, 29
loop, 7, 29
loop
method, 7, 12, 26, 29
method, 7, 12, 25, 26, 29
method, 7, 12, 25, 26, 29
method, 7, 12, 29
method, 7, 26, 29
method, 7, 29
method
options, 29
options, 26, 29
options
over, 26
over, 26
over
public, 26, 29
public, 26, 29
public
public
rexx, 7, 29
rexx, 7, 29
rexx
say, 25
say, 25
say
signal, 12
signal, 12
signal
to, 25, 29
to
where, 31
where, 31
where