

Pipelines Guide and Reference

Ed Tomlinson

Jeff Hennick

René Jansen

Version 4.02-GA of January 24, 2022

THE REXX LANGUAGE ASSOCIATION
NetRexx Programming Series
ISBN 978-90-819090-3-7

Publication Data

©Copyright The Rexx Language Association, 2011- 2022

All original material in this publication is published under the Creative Commons - Share Alike 3.0 License as stated at <http://creativecommons.org/licenses/by-nc-sa/3.0/us/legalcode>.

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

This edition is registered under ISBN 978-90-819090-3-7

ISBN 978-90-819090-3-7



9 789081 909037 >

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

14	Giving commands to the operating system	28
14.1	Built-ins	28
15	TCP/IP Networking	29
16	Selecting from relational databases	31
17	The Pipes Runner	32
18	The Pipes Compiler	33
19	Built-in Stages	34
20	Differences with CMS Pipelines	35
	Index	83

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.

Quick Start Guide

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.

Programming Guide

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.

Language Reference

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.

Pipelines Guide & Reference

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.

Introduction

A Pipeline, or Hartmann Pipeline¹², 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*.

¹https://en.wikipedia.org/wiki/CMS_Pipelines

²This 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

³In 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 16 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.

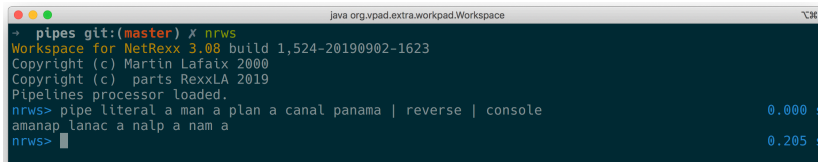


FIGURE 1: Run in the NetREXX Workspace

```
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.

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

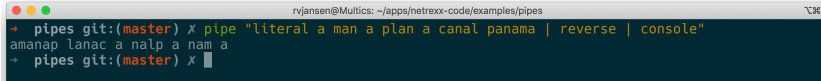


FIGURE 2: Run from the OS command line

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

3.4 Precompiled Pipelines

In this mode, which uses the `picp` 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.

```
picp "(test1) literal a man a plan a canal panama | reverse | console"
```

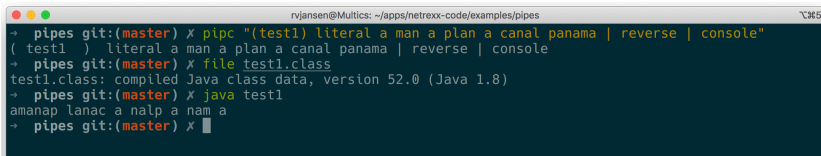


FIGURE 3: Precompile a Pipeline from the OS command line

This will yield a

```
test1.class
```

classfile, which can be executed by the java virtual machine.

The file `test1.class` can be run with the command⁴:

```
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:

⁴or 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

```
nIWS
```

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

```
pipe command ls -laFTl | console
```

```

test git:(master) X nrws
Workspace for NetRexx 3.08 build 1,524-20190902-1623
Copyright (c) Martin Lafaix 2000
Copyright (c) parts RexxLA 2019
Pipelines processor loaded.
nrws> pipe command ls -laFTl | console
total 17256
drwxr-xr-x 254 rvjansen staff 8128 Sep 2 17:09:12 2019 ./
drwxr-xr-x+ 183 rvjansen staff 5856 Sep 2 17:07:54 2019 ../
-rw-r--r--@ 1 rvjansen staff 6148 Nov 5 10:30:11 2018 .DS_Store
drwxr-xr-x 16 rvjansen staff 512 Sep 2 17:07:51 2019 .git/
-rw-r--r-- 1 rvjansen staff 9 Oct 9 14:28:25 2017 .gitignore
-rw-r--r-- 1 rvjansen staff 392 Oct 9 14:28:25 2017 AddFile.nrx
-rw-r--r-- 1 rvjansen staff 784 Oct 9 14:28:25 2017 BaseChange.nrx
-rw-r--r-- 1 rvjansen staff 208 Oct 9 14:28:25 2017 Calculator.nrx
-rw-r--r-- 1 rvjansen staff 223 Oct 9 14:28:25 2017 CalculatorTest.nrx
-rw-r--r--@ 1 rvjansen staff 1580 Dec 12 09:28:37 2017 ChangeFile1.nrx
-rw-r--r--@ 1 rvjansen staff 871 Dec 12 09:28:37 2017 ChangeFile2.nrx
-rw-r--r--@ 1 rvjansen staff 1772 Dec 12 09:28:37 2017 ChangeFile3.nrx
-rw-r--r-- 1 rvjansen staff 897 Oct 9 14:28:25 2017 ChangeReport.nrx
-rw-r--r-- 1 rvjansen staff 228 Oct 9 14:28:25 2017 ChangedFiles.nrx
-rw-r--r-- 1 rvjansen staff 1430 Jul 19 20:26:21 2018 CompilerVersion.java
-rw-r--r-- 1 rvjansen staff 1280 Oct 22 16:39:19 2018 DownloadFile.nrx
-rw-r--r-- 1 rvjansen staff 3136 Dec 15 10:11:36 2018 DownloadPDS.nrx
-rw-r--r-- 1 rvjansen staff 4647 Jul 20 17:13:39 2018 DynamicHelloWorld.java
-rw-r--r-- 1 rvjansen staff 570 Jul 22 00:20:31 2018 ECDCompilerVersion.nrx
-rw-r--r-- 1 rvjansen staff 575 Oct 9 14:28:25 2017 EbdictTest.java
-rw-r--r-- 1 rvjansen staff 1354 Aug 7 09:14:04 2019 EnzoFX1.class
-rw-r--r-- 1 rvjansen staff 2997 Aug 7 09:14:04 2019 EnzoFX.class
-rw-r--r-- 1 rvjansen staff 2432 Aug 7 07:04:21 2019 EnzoFX.java
-rw-r--r-- 1 rvjansen staff 1402 Aug 7 10:23:18 2019 EnzoLCD$timer_.class
-rw-r--r-- 1 rvjansen staff 3282 Aug 7 10:23:18 2019 EnzoLCD.class
-rw-r--r-- 1 rvjansen staff 2142 Aug 8 06:46:40 2019 EnzoLCD.nrx
-rw-r--r-- 1 rvjansen staff 601 Oct 9 14:28:25 2017 Expr.g4
-rw-r--r-- 1 rvjansen staff 1258 Oct 9 14:28:25 2017 ExprJoyRide.java
-rw-r--r-- 1 rvjansen staff 413 Jul 24 08:45:05 2019 FormatNumber.nrx
-rw-r--r-- 1 rvjansen staff 1116 Dec 15 05:23:52 2014 FtpPDS.nrx

```

FIGURE 4: example 1

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

```
java org.yypad.extra.workpad.Workspace
nrws> pipe command ls -laFTl | rexx specs 42-47 1 58-* 8 | console 0.291 s
Sep  2 2019 ./
Sep  2 2019 ./
Nov  5 2018 .DS_Store
Sep  2 2019 .git/
Oct  9 2017 .gitignore
Oct  9 2017 AddFile.nrx
Oct  9 2017 BaseChange.nrx
Oct  9 2017 Calculator.nrx
Oct  9 2017 CalculatorTest.nrx
Dec 12 2017 ChangeFile1.nrx
Dec 12 2017 ChangeFile2.nrx
Dec 12 2017 ChangeFile3.nrx
Oct  9 2017 ChangeReport.nrx
Oct  9 2017 ChangedFiles.nrx
Jul 19 2018 CompilerVersion.java
Oct 22 2018 DownloadFile.nrx
Dec 15 2018 DownloadPDS.nrx
Jul 20 2018 DynamicHelloWorld.java
Jul 22 2018 ECJCompilerVersion.nrx
Oct  9 2017 EbdicTest.java
Aug  7 2019 EnzoFX$1.class
Aug  7 2019 EnzoFX.class
Aug  7 2019 EnzoFX.java
Aug  7 2019 EnzoLCD$timer_.class
Aug  7 2019 EnzoLCD.class
Aug  8 2019 EnzoLCD.nrx
Oct  9 2017 Expr.g4
Oct  9 2017 ExprJoyRide.java
Jul 24 2019 FormatNumber.nrx
Dec 15 2014 FtpPDS.nrx
Oct  9 2017 GetSha1.nrx
Oct  9 2017 HelloWorldMainTopic.java
Oct  9 2017 HexPrint.nrx
Mar 28 2018 IMG_0725_2.jpg
Mar 28 2018 IMG_0735_2.jpg
```

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
```



```

java org.vpad.extra.workpad.Workspace
nrws> pipe command ls -laFTl | rexx specs 42-47 1 58-* 8 | sort | console 0.186 s
Apr  2 2019 readZip.nrx
Apr  5 2019 testFile.nrx
Apr  8 2019 listZip.nrx
Apr 10 2019 a.test
Apr 10 2019 b.test
Apr 10 2019 unzip.nrx
Apr 12 2018 testTextLine.nrx
Apr 24 2018 hello.c
Apr 24 2018 hello.s
Aug  5 2019 clg.class
Aug  5 2019 clg.nrx
Aug  7 2019 EnzoFX$1.class
Aug  7 2019 EnzoFX.class
Aug  7 2019 EnzoFX.java
Aug  7 2019 EnzoLCD$timer_.class
Aug  7 2019 EnzoLCD.class
Aug  7 2019 R$Anrx.class
Aug  7 2019 R$Anrx.nrx
Aug  7 2019 R$Anrx.nrx
Aug  7 2019 TreeViewSample.class
Aug  7 2019 TreeViewSample.java
Aug  7 2019 TreeViewSampleNRX.class
Aug  7 2019 TreeViewSampleNRX.nrx
Aug  7 2019 testLoop100miljoen.cpp
Aug  7 2019 testLoop100miljoenNRX.class
Aug  7 2019 testLoop100miljoenNRX.nrx
Aug  7 2019 testLoop100miljoen_JAVA.class
Aug  7 2019 testLoop100miljoen_JAVA.java
Aug  7 2019 testMainTopic$buttonAction.class
Aug  7 2019 testMainTopic.class
Aug  8 2019 EnzoLCD.nrx
Aug 13 2018 bigdog.nrx
Aug 13 2018 dog.nrx
Aug 13 2019 dyn.class
Aug 13 2019 dyn.nrx
Aug 14 2018 iban.nrx

```

FIGURE 6: example 3

```

java org.vpad.extra.workpad.Workspace
nrws> pipe command ls -laFTl | rexx specs 42-47 1 58-* 8 | specs 7-11 1 1-6 7 12-* 12 | sort | console
2014 Dec 1 FtpPDS.nrx
2014 Dec 1 SubmitJob.nrx
2014 Dec 1 UploadFile.nrx
2014 Dec 1 UploadPDS.nrx
2016 Aug 1 xml/
2016 Sep 2 db2conn.rexx
2016 Sep 2 db2query.rexx
2016 Sep 2 invdsnutils.rexx
2016 Sep 2 rvjcmf01.panel
2016 Sep 2 rvjcmf01.rexx
2016 Sep 2 rvjcmf01.skel
2017 Dec 1 ChangeFile1.nrx
2017 Dec 1 ChangeFile2.nrx
2017 Dec 1 ChangeFile3.nrx
2017 Jun 2 kitchen_sink.bxml
2017 Oct .gitignore
2017 Oct AddFile.nrx
2017 Oct BaseChange.nrx
2017 Oct Calculator.nrx
2017 Oct CalculatorTest.nrx
2017 Oct ChangeReport.nrx
2017 Oct ChangedFiles.nrx
2017 Oct EccdtTest.java
2017 Oct Expr.g4
2017 Oct ExprJoyRide.java
2017 Oct GetSha1.nrx
2017 Oct HelloWorldMainTopic.java
2017 Oct HexPrint.nrx
2017 Oct InsertFile.nrx
2017 Oct JGitEBC.nrx
2017 Oct Migrate.nrx
2017 Oct MyPlayground.playground/
2017 Oct RSA.java
2017 Oct RetrieveFile.nrx

```

FIGURE 7: example 4

```

java org.vpad.extra.workpad.Workspace
nrws> pipe command ls -laFTl | rexx specs 42-47 1 58-* 8 | specs 7-11 1 1-6 7 12-* 12 | sort | locate /2019 Mar 2/ | console
2019 Mar 2 commandtest.njp
2019 Mar 2 testc2x.nrx
nrws>
0.157 s

```

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 example⁵), that you have an assignment to quickly reverse a string.

```
/* 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)

```
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))
```

⁵From 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.

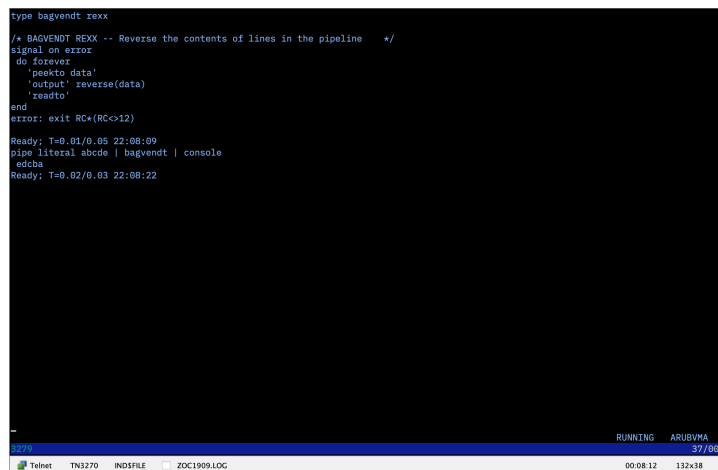
We can test this by building a pipeline and running the filter on its own source:

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

If you have a CMS handy, that would be:

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

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



```
type bagvendt rexx
/* 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)
Ready: T=0.01/0.05 22:08:09
pipe literal abcd | bagvendt | console
edcba
Ready: T=0.02/0.03 22:08:22
```

FIGURE 9: BAGVENDT under VM/CMS

```
ryjansen@Multics: ~/apps/netrexx-codes/examples/pipes
+ pipes git:(master) x cat bagvendt.nrx
import org.netrexx.njppipes.pipes.
class bagvendt extends stage
method run()
    loop forever
        line = Rexx peekto()
        output(line.reverse())
        readto()
    catch StageError
        rc = rc()
    end -- loop forever
    exit(rc*(rc<=12))
+ pipes git:(master) x nrc bagvendt
NetRexx portable processor 3.08-GA build 1,524-20190902-1623
Copyright (c) RexxIA, 2011,2019. All rights reserved.
Parts Copyright (c) IBM Corporation, 1995,2008.
Program bagvendt.nrx
=== class bagvendt ===
method run
    signals Thread0
    overrides stage.run
Compilation of 'bagvendt.nrx' successful
+ pipes git:(master) x pipe "literal abcde | bagvendt | console"
edcbe
+ pipes git:(master) x █
```

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 ⁶ implements the shell of a multitasking server - using about eight stages. The file `examples/tcptask.njp` contains an example of this technique being used.

```
--tasktest.njp
```

```
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 |
    elastic |                             -- buffer requests to so no deadlocks
    ptimer |

a: |
    copy |                               -- buffer work so no deadlocks
    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.

⁶using code from Melinda Varians 'Cramming for the Journeyman Plumber Exam' paper

Device Drivers

Pipelines for NetREXX contains the following device drivers:
(see next page)

<	read from a file
>	write to a file (which is overwritten if it exists)
»	append to a file (which is created if it does not exist)
diskr	read from a file
diskw	write to a file (which is overwritten if it exists)
diska	append to a file (which is created if it does not exist)
diskslow	read, create or append to a file
array	manipulate arrays
arraya	manipulate arrays
arrayr	manipulate arrays
stem	manipulate stems
stema	manipulate stems
stemr	manipulate stems
vector	manipulate vectors
vectora	manipulate vectors
vectorr	manipulate vectors
var	read or set a variable in a NetRexx program
zip	compress a set of files (0 or more) into a zip archive
unzip	decompress a set of files (0 or more) from a zip archive
listzip	list a zip file directory
console	read from, or write to a terminal (window)
hole	destroy data
delay	suspend stream
literal	write the argument string
strliteral	write the argument string
sqlselect	select from any jdbc source
xrange	write a character range

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:

between	selects records between labels
drop	discard records from the beginning or the end of a file
find	select lines
strfind	select lines
flabel	select records from the first one with leading string
strflabel	select records from the first one with leading string
inside	select records between labels
locate	select records between labels
nfind	select lines using xedit nfind logic
strnfind	select lines using xedit nfind logic
nlocate	select lines without a string
notinside	select records not between labels
outside	select records not between labels
pick	select records that satisfy a relation
take	select records from the beginning or the end of a file
tolabel	select records to the first one with leading string
strtolabel	select records to the first one with leading string
unique	discard or retain duplicate lines

Filters

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

Other Stages

query	check version and level of Pipelines for NetREXX
--------------	--------------------------------------------------

-- --	insert comments into a pipeline
--------------	---------------------------------

comment	insert comments into a pipeline
----------------	---------------------------------

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*⁷. 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:

⁷often, 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:
```

```

pipes git:(master) X nrws
Workspace for NetREXX 3.00 build 1,524-20190902-1623
Copyright (c) Martin Lafatz 2000
Copyright (c) parts RexxLA 2019
Pipelines processor loaded.
nrws> pipe literal test | a: fanin | console | a:          0.000 s
test
Deadlocked in p7f2dc97
Dumping p7f2dc97 Stall 2000 Monitored by p7f2dc97
Flag units digit: 1=wait out, 2=wait in, 4=wait any, 8=wait commit
                  : 10=pending autocommit, 20=pending sever

literal_1
Running rc=0 commit=-1 Flag=201 waits 0 args=test
-> out 0 fanin_2 1 test

fanin_2
Running rc=0 commit=-1 Flag=201 waits 0 args=
-> in 0 literal_1 1 test
-> in 1 console_3 0 test
-> out 0 console_3 1 test

console_3
Running rc=0 commit=-1 Flag=201 waits 0 args=
-> in 0 fanin_2 1 test
-> out 0 fanin_2 0 test

Dumped Pipe p7f2dc97 Flag 60F rc=16

RC=16
nrws> 4.205 s

```

FIGURE 11: Deadlock detection

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:

```

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 *n*th 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-

plete.

```
-- over.nrx
class over extends stage final

  method run() public
    a = getRexx(arg())
    loop i over a
      output(a[i])
    catch StageError
      rc = rc()
    end

  exit(rc*(rc<>12))
```

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:

```
-- 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

  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:
```



```
*inN:  
*out:  
*outN
```

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 stages'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, append 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.

```
-- 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
  onerresponse |
  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 onerresponse |
      elastic |
      insert /\n/ after |
    f:

  catch StageError
    rc = rc()
end
```

```
exit(rc*(rc<>12))
```

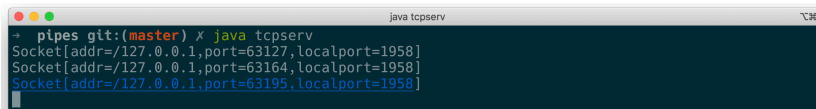
This example needs to be compiled with the pipes compiler, see *TCP/IP Client/Server compile*, which yields the classes `tcpserv` and `tcpreq`, for the server and the requester component.



```
rvjansen@Multics: ~/apps/netrexx-code/examples/pipes
→ pipes git:(master) x pipc tcpexample.njp
pipe (tcpserv stall 60000 debug 0 ) tcplisten 1958 timeout 15000 | tcpexample
pipe (tcpreq stall 60000 debug 0 ) random arg() | specs *-* 1 ,\n, next | tcpclient deblock c loc
alhost 1958 timeout 10000 linger 500 onerresponse | rexx to console
callpipe (tcpexample_tcplog stall 15000 debug 0) *i_A0: | take first 1 | console | f: fanin | tcp
data timeout 10000 deblock C onerresponse | elastic | insert /\n/ after | f:
→ pipes git:(master) x
```

FIGURE 12: TCP/IP Client/Server compile

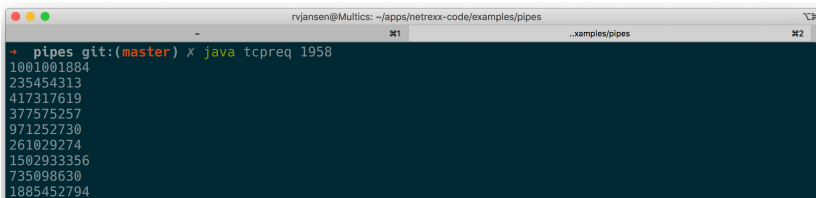
Now we can start the generated pipelines each in their own shell window. As can be seen in *TCP/IP server*, the class keeps waiting on connections on port 1958 - which is arbitrary, but specified in the pipeline source.



```
java tcpserv
→ pipes git:(master) x java tcpserv
Socket[addr=/127.0.0.1,port=63127,localport=1958]
Socket[addr=/127.0.0.1,port=63164,localport=1958]
Socket[addr=/127.0.0.1,port=63195,localport=1958]
```

FIGURE 13: TCP/IP server

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.



```
rvjansen@Multics: ~/apps/netrexx-code/examples/pipes
→ pipes git:(master) x java tcpreq 1958
1001001884
235454313
417317619
377575257
971252730
261029274
1502933356
735098630
1885452794
```

FIGURE 14: TCP/IP requestor

Note that the stage `tcpexample` from the `tcpserv` 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 commandline 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 persistence⁸.

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 macOS⁹.

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:

pipename	Specify the name of the generated class file. This can be useful for debugging purposes but is not mandatory when running a pipe. An unnamed pipe receives a generated unique name. This option needs to go first.
sep	The default stage separator is the (pipe) character; this can be overridden with the sep option; a pipe called test1 which uses an exclamation mark as separator character, needs the options (test1 sep !).
debug	The debug option specifies a bitmask for debugging the execution of a pipe; (debug 63), for example, generates a rather complete debugging trail).
end	The default pipe end character is the '?' (question mark), which can be overridden here. Note that the backslash, which is an obvious pipe end character for the z/VM 3270 interface, is not a good choice for Windows and Unix shells.
stall	The duration in number of seconds of a pipe stall (or deadlock) detection cycle.

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:

-gen Generate the NetREXX source file. The pipeline needs a name.

-keep Keep the Java source which is generated from the NetREXX source.

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 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 similiar 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 Builtin Stages

Show Stages Implemented in:

NetRexx Pipelines: CMS Pipelines:

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

Highlight NetRexx Only / CMS Only:

<p>> <i>diskw</i> <i>filew</i> 3.09</p>	<p>Replace or Create a File</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>►--->---string---◀</p> </div> <ul style="list-style-type: none"> • delegates to diskw.
<p>>> <i>diska</i> <i>filea</i> 3.09</p>	<p>Append to or Create a File</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>►--->>---string---◀</p> </div> <ul style="list-style-type: none"> • delegates to diska.
<p>>><i>mdsk</i></p>	<p>Append to or Create a CMS File on a Mode</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p>>><i>mvs</i></p>	<p>Append to a Physical Sequential Data Set</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p>>><i>oe</i></p>	<p>Append to or Create an OpenExtensions Text File</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p>>><i>sfs</i></p>	<p>Append to or Create an SFS File</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p>>><i>sfs slow</i></p>	<p>Append to or Create an SFS File</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p>><i>mdsk</i></p>	<p>Replace or Create a CMS File on a Mode</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

>mvs	Rewrite a Physical Sequential Data Set or a Member of a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>oe	Replace or Create an OpenExtensions Text File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>sfs	Replace or Create an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
< diskr disk file filer 3.09	Read a File <div data-bbox="311 443 576 499" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> ▶▶---<--string--▶▶ </div> <ul style="list-style-type: none"> • Implemented as in CMS; delegates to diskr.
<m disk	Read a CMS File from a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<mvs	Read a Physical Sequential Data Set or a Member of a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<oe	Read an OpenExtensions Text File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<sfs	Read an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<sfsslow	Read an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
-- comment 3.09	Comment Stage, No Operation <div data-bbox="311 1218 842 1295" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> NetRexx ▶▶---+- -- ----+---+-----+-----▶▶ +-COMMENT-+ +-string-+ </div> <ul style="list-style-type: none"> • delegates to comment. • 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 used before a driver stage, it must have a pipeEnd.
3277bfra	Convert a 3270 Buffer Address Between Representations <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
3277enc	Write the 3277 6-bit Encoding Vector <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

<p><i>64decode</i> <i>decode64</i> 3.11</p>	<p>Decode Base-64 Format</p> <div data-bbox="309 121 772 254" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre> +-MIME--+ ▶---+-64DECODE-+---+-----+---▶ +-BASIC-+ +-URL---+ +-DECODE64-+ </pre> </div> <ul style="list-style-type: none"> 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 ou 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 eac uses a carriage return 'r' followed by a linefeed 'n' as the line separator. No line separator is present to the end of t encoded output. 3.11: New to NetRexx. Add MIME, BASIC, & URL options.
<p><i>64encode</i> <i>encode64</i> 3.11</p>	<p>Encode to Base-64 Format</p> <div data-bbox="309 611 772 743" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre> +-MIME--+ ▶---+-64ENCODE-+---+-----+---▶ +-BASIC-+ +-URL---+ +-ENCODE64-+ </pre> </div> <ul style="list-style-type: none"> 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 ou 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 eac uses a carriage return 'r' followed by a linefeed 'n' as the line separator. No line separator is present to the end of t encoded output. 3.11: New to NetRexx. Add MIME, BASIC, & URL options.
<p>? <i>help</i></p>	<p>Display Help for Pipelines</p> <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<p><i>abbreviation</i> <i>abbreviatio</i> <i>abbreviati</i> <i>abbreviat</i> <i>abbrevi</i> <i>abbrev</i></p>	<p>Select Records that Contain an Abbreviation of a Word in the First Positions</p> <div data-bbox="309 1167 1182 1402" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre> ▶---ABBREViation- (1) -+-----+---▶ +-word-+-----+---+ +-number-+-----+---+ +-ANYcase- (2) -+ +-CASEANY-----+ +-CASEIGNORE---+ +-IGNORECASE---+ +-CASELESS-----+ </pre> </div> <ul style="list-style-type: none"> (1) ABBREViation must be ABBREV in CMS (2) ANYcase must be ANYCASE in CMS
<p><i>acigroup</i></p>	<p>Write ACI Group for Users</p> <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<p><i>addrdw</i></p>	<p>Prefix Record Descriptor Word to Records</p> <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.

<i>adrspc</i>	Manage Address Spaces <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>aftfst</i>	Write Information about Open Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>aggrc</i>	Compute Aggregate Return Code <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 5px 0;"> ▶▶--AGGRC--◀◀ </div>
<i>all</i>	Select Lines Containing Strings (or Not) <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>alserv</i>	Manage the Virtual Machine's Access List <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>apldecode</i>	Process Graphic Escape Sequences, Old APL language <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>aplencode</i>	Generate Graphic Escape Sequences, Old APL language <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>append</i>	Put Output from a Device Driver after Data on the Primary Input Stream <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 5px 0;"> ▶▶--APPEND--string-----▶▶ </div>
<i>array</i>	Read or Write an Array <ul style="list-style-type: none"> • Pipes for NetRexx
<i>arraya</i>	Read or Write an Array <ul style="list-style-type: none"> • Pipes for NetRexx
<i>arrayr</i>	Read or Write an Array <ul style="list-style-type: none"> • Pipes for NetRexx
<i>arrayw</i>	Read or Write an Array <ul style="list-style-type: none"> • Pipes for NetRexx
<i>asatomc</i>	Convert ASA Carriage Control to CCW Operation Codes. Old printer control <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>asmcont</i>	Join Multiline Assembler Statements <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>asmfind</i>	Select Statements from an Assembler File as XEDIT Find <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>asmnfind</i>	Select Statements from an Assembler File as XEDIT NFind <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>asmxpnd</i>	Expand Joined Assembler Statements <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>beat</i>	Mark when Records Do not Arrive within Interval <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

<p><i>between</i> 3.09</p>	<p>Select Records Between Labels</p> <pre> ▶▶--BETWEEN-- case --<u>delimitedString1</u>--+-number-----+--▶▶ +-Xhexstring1-----+ +-delimitedString2--+ +-Hhexstring1-----+ +-Xhexstring2-----+ +-Bbinstring1-----+ +-Hhexstring2-----+ +-Bbinstring2-----+ case: -----+ +-ANYcase-----+ +-CASEANY-----+ +-CASEIGNORE--+ +-IGNORECASE--+ +-CASELESS-----+ </pre>
<p><i>block</i></p>	<p>Block to an External Format</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>browse</i></p>	<p>Display Data on a 3270 Terminal</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>buffer</i></p>	<p>Buffer Records</p> <pre> ▶▶--BUFFER--+-----+-----+-----▶▶ +-number--+-----+-----+ +-delimitedString+ </pre>
<p><i>c14to38</i></p>	<p>Combine Overstruck Characters to Single Code Point. Old printer.</p>
<p><i>casei</i></p>	<p>Run Selection Stage in Case Insensitive Manner</p> <pre> ▶▶--CASEI--+-----+-----+-----▶▶ +-ZONE-- inputRange -(1) +-REVERSE-(1) ▶▶--stage+-----+-----▶▶ +-string+ </pre> <ul style="list-style-type: none"> • (1) CMS Pipelines only.

```

▶--CHANGE--+-----+-----+-----▶
      +-| anycase |-+ +-inputRange-----+
                          |-----+ |
                          +-(---range+---)-+

      +-FROMTO-(1)+
▶--+-----+---| changeString |-----+---▶
  | +-FROM-(1)+          +-TO-(1)+          |
  +-+-----+delimitedString+-----+delimitedString+

▶--+-----+-----+-----▶◀
  | +-FOR-(1)+          |
  +-+-----+numorstar+

```

```

changeString:
|--delimiter--string--delimiter--string--delimiter--|

```

```

anycase:
+-RESPECTCASE-(1)+
|--+-----+---|
+-ANYcase-----+
+-CASEANY-----+
+-CASEIGNORE-----+
+-IGNORECASE-----+
+-CASSLESS-----+

```

- (1) NetRexx Pipelines only.

changeparse
 changepars
 changepar
 changepa
 changep
 3.11

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-+ +-FINALLY-finally template 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. It may contain 0 or more semicolon separated NetRexx statements, probably using the \$n variables. 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 won't be changed./
- changeparse from / 2 \$1 +1/ to /The second letter is "\$1". \$\$1 won't be changed./
- changeparse from / . \$2 . 50 \$5 +5 / to /The product is \$1/ by /\$1 = \$2 * \$5/
- changeparse from / . \$2 . 50 \$5 +5 / ,
 to /The product is \$1/ ,
 by /\$1 = \$2 * \$5;\$3 = \$3 + \$1/ ,
 first /\$3 = 0/ ,
 finally /\$3 is the total/

changeregex
 changerege
 changereg
 changere
 changer
 3.09

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
 truncat
 trunca
 trunc
 3.09

Truncate the Record

```

      +---80-----+
▶---+---CHOP---+-----+-----+-----+-----+-----+-----+-----+-----+-----▶
      +-TRUNCate-+ +-snumber-----|
      +---| stringtarget |--+
stringtarget:
|-----+-----+-----+-----+-----+-----+-----+-----+-----| target |-|
+-ANYcase-----+ |           +-BEFORE-+ | +-NOT-+
+-CASEANY-----+ +-----+-----+-----+-----+
+-CASEIGNORE---+ +-snumber-+ +-AFTER---+
+-IGNORECASE---+
+-CASELESS-----+

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

cipher

Encrypt and Decrypt Using a Block Cipher

- Not implemented in Netrex Pipelines.

<i>ckddebblock</i>	Deblock Track Data Record <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>cmd</i> <i>command</i>	Issue OS Commands, Write Response to Pipeline <div data-bbox="309 226 957 306" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>▶---+--CMD-----+--+-----+-----▶ +-COMMAND-+ +--string--+</pre> </div> <ul style="list-style-type: none"> • 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
<i>cms</i>	Issue CMS Commands, Write Response to Pipeline
<i>collate</i> 3.11	Collate Streams <div data-bbox="309 577 1155 1155" style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre>▶---+-----+-----+-----+-----▶ +-NOPAD-----+ +-STOP ANYOF-+ +-PAD-xorc-+ +- anycase --+ +-MASTER DETAIL---+ +-inputRange-+-----+ +-MASTER-----+ +-inputRange-+ +-DETAIL-+-----+ +-MASTER-+ +-SEPARATOR-delimitedString- (1) (2) -+ anycase: +-RESPECTCASE- (1) + --+-----+--- +-ANYcase-----+ +-CASEANY-----+ +-CASEIGNORE-----+ +-IGNORECASE-----+ +-CASSLESS-----+</pre> </div> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines. • (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.
<i>combine</i>	Combine Data from a Run of Records <div data-bbox="309 1428 1184 1690" style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre>▶---COMBINE---+-----+-----+-----+-----▶ +-1-----+ +-aNd-----+ +-+-----+ +-AND-----+ +-number-----+ +-eXclusiveor-+ +-*------+ +-EXclusiveor-+ +-KEYLENgth--number-+ +-FIRST- (2) ----+ +-ALLEOF- (1)+ +-LAST- (2) ----+ +-STOP- (1) -+ +-+ +-ANYEOF- (1) +</pre> </div> <ul style="list-style-type: none"> • (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.

<p><i>command</i> <i>cmd</i></p>	<p>Issue OS Commands, Write Response to Pipeline</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>▶---+-COMMAND-+---+-----+-----▶ +-CMD-----+ +--string--+</pre> </div> <ul style="list-style-type: none"> • 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
<p><i>comment</i> -- 3.09</p>	<p>Comment stage</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>NetRexx</p> <pre>▶---+---COMMENT-+---+-----+-----▶ + -- -----+ +--string--+</pre> </div> <ul style="list-style-type: none"> • 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.
<p><i>compare</i></p>	<p>Compare Primary and Secondary Streams, Write the Result</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>NetRexx</p> <pre>▶---+---COMPARE-+---+-----+-----▶ +-TRINARY-+ (1) +-PAD SPACE-+ +-BINARY--+ (2) +-PAD-xorc--+ +-ECHO-+ -----+ -----+ +--ANY delimitedString-----+---+ (4) (5) +--EQUAL delimitedString-----+ (4) +--LESS delimitedString-----+ (3) (4) +--MORE delimitedString-----+ (3) (4) +--NOTEQUAL delimitedString-+ (4)</pre> </div> <ul style="list-style-type: none"> • (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) <i>DStrings</i> can use any of the following escapes (or the lowercase) for the unequal situation: <ul style="list-style-type: none"> ◦ \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 <i>DString</i> 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: <ul style="list-style-type: none"> ◦ 0: Data 1 ◦ 1: Data 2 • Output streams: <ul style="list-style-type: none"> ◦ 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
<p><i>configure</i></p>	<p>Set and Query CMS Pipelines Configuration Variables</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.


```

▶--DATECONvert--+-----+-----▶
                    +--inputRange- (3) -+

  +-SHOrtdate ISOdate-----+ +-WINDOW -50-----+
▶+-----+-----+-----+-----▶
  |                               +-ISOdate-----+ +-WINDOW-signednumber-+
+-| Inputformat |+-+-----+ +-BASEYEAR-yearnumber-+
  |   +-PREFACE-+ | +-| Outputformat |+-+
+-NOW-+- (5) -----+
      +-APPEND--+

                               +--MIDNIGHT-- (4) -+
▶+-----+-----+-----+-----▶
      +--TIMEOUT--+ +--NOON-- (4) -----+

```

Inputformat,	Outputformat:
SHOrtdate } USA_SHORT } REXX_DATE_U }	mm/dd/yy hh:mm:ss.uuuuuu
FULldate } USA }	mm/dd/yyyyyy hh:mm:ss.uuuuuu
ISO_SHORT ISOdate DB2_SHORT DB2 VMDATE (2) NORMAL CSL_SHORT } REXX_DATE_O } CSL PIPE_SHORT PIPE } REXX_DATE_S } EURSHORT EUR JULIAN_SHORT JULIAN	yy-mm-dd hh:mm:ss.uuuuuu yyyyyy-mm-dd hh:mm:ss.uuuuuu yy-mm-dd-hh.mm.ss.uuuuuu yyyyyy-mm-dd-hh.mm.ss.uuuuuu dd mmm yyyyyyy hh:mm:ss.uuuuuu yy/mm/dd hh:mm:ss.uuuuuu yyyyyy/mm/dd hh:mm:ss.uuuuuu yymmddhhmmssuuuuuu yyyyymmddhhmmssuuuuuu dd.mm.yy hh:mm:ss.uuuuuu dd.mm.yyyyyyy hh:mm:ss.uuuuuu yy.ddd hh:mm:ss.uuuuuu yyyyyy.ddd hh:mm:ss.uuuuuu
TOD_ABSOLUTE } (2) TODABS } (2) SCIENTIFIC_ABSOLUTE } (2) SCIABS } (2) POSIX TOD_RELATIVE } (2) TODREL } (2) SCIENTIFIC_RELATIVE } (2) SCIREL } (2) MET (2)	ssssssss

The following can be REXX_DATE_x, REXXx, or Rx

```

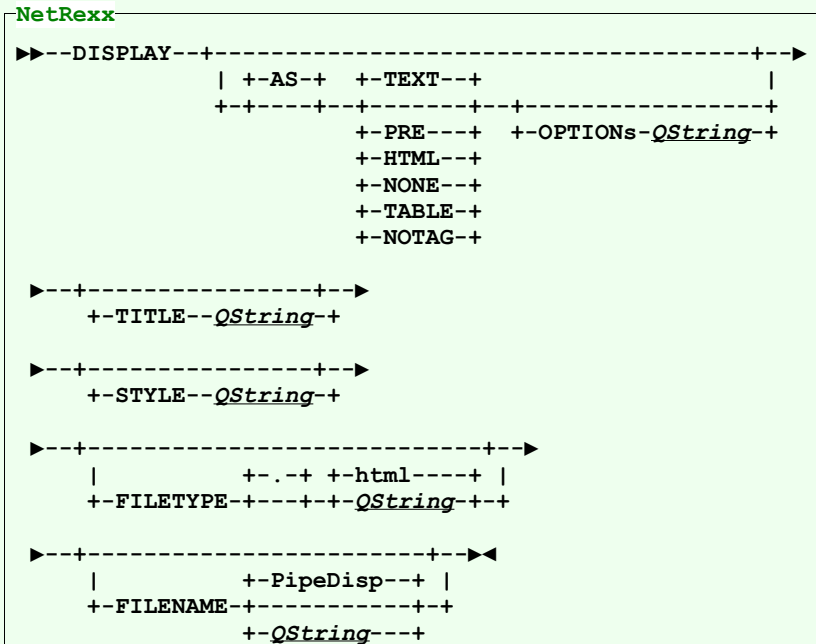
REXX_DATE_B (2)
REXX_DATE_C (2)
REXX_DATE_D ddd hh:mm:ss.uuuuuu
REXX_DATE_E dd/mm/yy hh:mm:ss.uuuuuu
REXX_DATE_E_LONG dd/mm/yyyyyy hh:mm:ss.uuuuuu
REXX_DATE_J yyddd hh:mm:ss.uuuuuu
REXX_DATE_J_LONG yyyyddd hh:mm:ss.uuuuuu
REXX_DATE_M mmmmmmmmm (output only)
REXX_DATE_N_SHORT dd mmm yy hh:mm:ss.uuuuuu
REXX_DATE_N dd mmm yyyy hh:mm:ss.uuuuuu
REXX_DATE_W wwwwwwww (output only)

```

- (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.

<i>deal</i>	<p>Pass Input Records to Output Streams Round Robin</p> <pre> +-STOP--ALLEOF-----+ ▶--DEAL--+-STOP--+-ALLEOF- (2) +-----+ +-ANYEOF-----+ +-number-----+ +-SECONDARY--+-RELEASE---+ +-LATCH- (1) -+ +-KEY--inputRange-+-STRIP-+ +-STREAMid--inputRange-+-STRIP-+ </pre> <ul style="list-style-type: none"> • (1) Not yet in NetRexx Pipelines • (2) Not CMS • 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.
<i>deblock</i>	<p>Deblock External Data Formats</p> <pre> +-80-----+ +-FIXED--+-number-+-+-----+-----+ +--PAD--xorc--+ (1) ▶--DEBLOCK--+-+-----+-----+-----+-----+-----+▶ +-+C-----+-----+-----+-----+-----+ +-J-----+-----+ +-TERMINATE+ +-EOF-+ +-CRLF-----+ +-LINEND xorc-----+ +-STRING--delimitedString-+ </pre> <ul style="list-style-type: none"> • CMS has many more mainframe centric formats that NetRexx Pipelines does not process. • (1) Not CMS Pipelines
<i>decode64</i> <i>64decode</i> <i>3.11</i>	<p>Decode Base-64 Format</p> <p>NetRexx</p> <pre> +-MIME--+ ▶--+-64DECODE-+-+-----+-----+▶ +-DECODE64-+ +-BASIC-+ +-URL-----+ </pre> <ul style="list-style-type: none"> • 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 ou 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 'r' followed by a linefeed 'n' as the line separator. No line separator is present to the end of t encoded output. • 3.11: New to NetRexx. Add MIME, BASIC, & URL options.
<i>delay</i>	<p>Suspend Stream</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>devinfo</i>	<p>Write Device Information</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>dfsrt</i>	<p>Interface to DFSORT/CMS</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>diage4</i>	<p>Submit Diagnose E4 Requests</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>dict</i> <i>hash</i>	<p>Read or Write a Dictionary</p> <ul style="list-style-type: none"> • Pipes for NetRexx only.

<i>dicta</i> <i>hasa</i>	Read or Write a Dictionary <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>dictr</i> <i>hasr</i>	Read or Write a Dictionary <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>dictw</i> <i>hashw</i>	Read or Write a Dictionary <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>digest</i>	Compute a Message Digest <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>disk</i> <i>diskr</i> < <i>file</i> <i>filer</i>	Read a File <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> NetRexx ▶▶--DISK--string--◀◀ </div> <ul style="list-style-type: none"> • As in CMS, equivalent to diskr (Pipes for NetRexx Only) or <.
<i>diska</i> >> <i>filea</i>	Append to or Create a File <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> NetRexx ▶▶--DISKA--string--▶▶ </div>
<i>diskback</i> <i>fileback</i>	Read a File Backwards <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskfast</i> <i>filefast</i>	Read, Create, or Append to a File <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskid</i>	Map CMS Reserved Minidisk <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskr</i> <i>disk</i> < <i>file</i> <i>filer</i>	Read a File <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> NetRexx ▶▶--DISKR--string--▶▶ </div> <ul style="list-style-type: none"> • As in CMS, equivalent to diskr (Pipes for NetRexx Only) or <.
<i>diskrandom</i> <i>filerandom</i>	Random Access a File <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskslow</i> <i>fileslow</i>	Read, Create, or Append to a File <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskupdate</i> <i>fileupdate</i>	Replace Records in a File <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>diskw</i> <i>filew</i> >	Replace or Create a File <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶-->--string--▶▶ </div>
<i>display</i> 3.11	Output to Web Browser



- DISPLAY works similar to and as a replacement for CONSOLE for output. But instead of going to the terminal window 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 & and < characters to < so & 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 anyway. 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 a named stylesheet. The name may have relative path names, or be an absolute file name. If there are spaces, enclose in quotes.

Note: This officially should go into a HEAD section; here it won't be there. Most modern browsers will honor it anyway. If it is not honored as a tag, it will not show -- except in the NOTAG option. The file itself is copied from its state 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 that system, external to Pipelines, is set up to recognize it, for example, "JSP" or "PHP". A "dot" is optional; only one will work. 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. The relative path is based on the working directory. If no path is specified in the name, the system Temp directory, as defined 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.

<i>drop</i>	<p>Discard Records from the Beginning or the End of the File</p> <div data-bbox="309 300 1112 432" style="border: 1px solid black; padding: 5px;"> <pre> +-FIRST--+ +-1-----+ ▶--DROP--++-----+-----+-----+-----▶ +-LAST--+ +-snumber- (1)+ +-BYTES-+ +-*-----+ </pre> </div> <ul style="list-style-type: none"> (1) CMS: must be positive. NetRexx Pipelines: negative reverses FIRST/LAST, so DROP FIRST -3 is the same as DROP LAST 3.
<i>duplicate</i> <i>duplicat</i> <i>duplica</i> <i>duplic</i> <i>dupli</i> <i>dupl</i> <i>dup</i>	<p>Copy Records</p> <div data-bbox="309 585 885 743" style="border: 1px solid black; padding: 5px;"> <pre> +-1-----+ ▶--DUPLICATE- (1) -+-----+-----▶ +-number-+ +-*-----+ +- -1-----+ </pre> </div> <ul style="list-style-type: none"> (1) CMS is DUPLICat due to 8-character name limitation
<i>elastic</i>	<p>Buffer Sufficient Records to Prevent Stall</p> <div data-bbox="309 869 659 926" style="border: 1px solid black; padding: 5px;"> <pre> ▶--ELASTIC-----▶ </pre> </div>
<i>encode64</i> <i>64encode</i> 3.11	<p>Encode to Base-64 Format</p> <p>NetRexx</p> <div data-bbox="309 978 772 1110" style="border: 1px solid black; padding: 5px;"> <pre> +-MIME--+ ▶--+-64ENCODE-+-----+-----▶ +-ENCODE64-+ +-BASIC-+ +-URL--+ </pre> </div> <ul style="list-style-type: none"> 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 ou 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 eac uses a carriage return '\r' followed by a linefeed '\n' as the line separator. No line separator is present to the end of t encoded output. 3.11: New to NetRexx. Add MIME, BASIC, & URL options.
<i>eofback</i>	<p>Run an Output Device Driver and Propagate End-of-File Backwards</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<i>escape</i>	<p>Insert Escape Characters in the Record</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<i>fanin</i>	<p>Concatenate Streams</p> <div data-bbox="309 1656 831 1761" style="border: 1px solid black; padding: 5px;"> <pre> ▶--FANIN--++-----+-----▶ -----+ +----stream-+-+ </pre> </div>
<i>faninany</i>	<p>Copy Records from Whichever Input Stream Has One</p> <div data-bbox="309 1814 675 1873" style="border: 1px solid black; padding: 5px;"> <pre> ▶--FANINANY-----▶ </pre> </div>
<i>fanintwo</i>	<p>Pass Records to Primary Output Stream</p>

<i>fanout</i>	<p>Copy Records from the Primary Input Stream to All Output Streams</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <pre> +-STOP--ALLEOF-----+ ▶--FANOUT--+-----+-----▶ +-STOP--+-ANYEOF-----+ +-ALLOF-- (1) ---+ +-number-----+ </pre> </div> <ul style="list-style-type: none"> • (1) CMS only
<i>fanouttwo</i>	Copy Records from the Primary Input Stream to Both Output Streams
<i>fbaread</i>	<p>Read Blocks from a Fixed Block Architecture Drive</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fbawrite</i>	<p>Write Blocks to a Fixed Block Architecture Drive</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fblock</i>	<p>Block Data, Spanning Input Records</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <pre> ▶--FBLOCK--number--+-----+-----▶ +-xorc--+ </pre> </div>
<i>file</i> <i>filer</i> <i>disk</i> <i>diskr</i>	<p>Read a File</p> <p>NetRexx</p> <div style="border: 1px solid black; padding: 2px; text-align: center;"> <pre>▶--FILE--string--▶</pre> </div>
<i>filea</i> <i>diska</i> >>	<p>Append to or Create a File</p> <div style="border: 1px solid black; padding: 2px; text-align: center;"> <pre>▶--FILEA--string--▶</pre> </div>
<i>fileback</i> <i>diskback</i>	<p>Read a CMS file backwards</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>filedescriptor</i>	<p>Read or Write an OpenExtensions File that Is Already Open</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>filefast</i> <i>diskfast</i>	<p>Read or write a CMS file</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>filer</i> <i>file</i> <i>disk</i> <i>diskr</i> <	<p>Read a File</p> <p>NetRexx</p> <div style="border: 1px solid black; padding: 2px; text-align: center;"> <pre>▶--FILER--string--▶</pre> </div>
<i>filerandom</i> <i>diskrandom</i>	Read specific records from a CMS file
<i>fileslow</i> <i>diskslow</i>	<p>Read, Create, or Append to a File</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>filetoken</i>	<p>Read or Write an SFS File That is Already Open</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fileupdate</i> <i>diskupdate</i>	Change records in a CMS file
<i>filew</i> <i>diskw</i> >	<p>Replace or Create a File</p> <p>NetRexx</p> <div style="border: 1px solid black; padding: 2px; text-align: center;"> <pre>▶--FILEW--string--▶</pre> </div>
<i>fillup</i>	<p>Pass Records To Output Streams</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

<i>filterpack</i>	Manage Filter Packages <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>find</i>	Select Lines by XEDIT Find Logic <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶-----FIND---+-----+-----▶▶ +--string--+ </div>
<i>fitting</i>	Source or Sink for Copipe Data <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>fmtfst</i>	Format a File Status Table (FST) Entry <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>fmlabel</i> <i>fromlabel</i>	Select Records from the First One with Leading String <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶-----FRLABEL--+-----+-----▶▶ +--string--+ </div>
<i>fromlabel</i> <i>fmlabel</i>	Select Records from the First One with Leading String <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶-----FROMLABEL--+-----+-----▶▶ +--string--+ </div>
<i>frtarget</i>	Select Records from the First One Selected by Argument Stage <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶---+--FRTARGET---+--stage--+-----+-----▶▶ +--FROMTARGET--+ +--operands--+ </div>
<i>fullscrq</i>	Write 3270 Device Characteristics <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>fullscrs</i>	Format 3270 Device Characteristics <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>gate</i>	Pass Records Until Stopped <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶--GATE--+-----+-----▶▶ +--STRICT--+ </div>
<i>gather</i>	Copy Records From Input Streams <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>getfiles</i> <i>getfiles</i> <i>getfile</i> <i>getfil</i> <i>getfi</i> <i>getf</i> <i>get</i>	Read Files <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶--GETfiles-----▶▶ </div>
<i>getovers</i>	Write the Contents of Objects <ul style="list-style-type: none"> Input stream 0 should contain rexx objects. The getovers stage will output the index and contents of the stem on str output stream 1 is connected, the root is placed there. Any severed streams will cause then stage to exit. Passing a object will cause the stage to exit with return code 13. Pipes for NetRexx only.
<i>getstems</i>	Write the Contents of Members of Stems <ul style="list-style-type: none"> Input stream 0 should contain rexx objects containing stems. The getstems stage will output the contents of the stem on stream 0. If output stream 1 is connected, the root is placed there. Any severed streams will cause then stage to exit. Passing a non rexx stem object will cause the stage to exit with return code 13. Pipes for NetRexx only.

grep
regex
3.09

Select Lines by a Regular Expression

NetRexx

```
▶---+---GREP---+---+-----+---regex_Dstring- (1)---▶◀
  +---REGEX---+  +---(--| options_string |---)-+

options_string:
  +-----+
|---+-----+---+-----+---+-----+
  +-Numbers-----+ (2)
  +-Before+-1-----+ (3)
  |           +-number-+           |
  +-After+-1-----+ (3)
  |           +-number-+           |
  +-Context+-1-----+ (4)
  |           +-number-+           |
  +-NOseparator-----+ (5)
  +-Separator+---/---/-----+ (5)
  |           +-delimitedString-+ |
  +-Tertiary-----+ (6)
  +-COUnt-----+ (7)
```

- 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

hash
dict

Read or Write a Dictionary

- Pipes for NetRexx only.

hasha
dicta

Read or Write a Dictionary

hashr
dictr

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.

<i>hfsquery</i>	Write Information Obtained from OpenExtensions into the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hfsreplace</i>	Replace the Contents of a File in the Hierarchical File System <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hfsstate</i>	Obtain Information about Files in the Hierarchical File System <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hfsxecute</i>	Issue OpenExtensions Requests <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hiasm</i>	Interface to High Level Assembler <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hiasmerr</i>	Extract Assembler Error Messages from the SYSADATA File <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>hole</i>	Destroy Data <pre>▶▶--HOLE-----▶▶</pre>
<i>hostbyaddr</i> 3.09	Resolve IP Address into Domain and Host Name <pre>▶▶-----HOSTBYADDR -----+-----+-----▶▶ +--INCLUDEIP--+ (1)</pre> <ul style="list-style-type: none"> (1) Optional parameter not present in VM/CMS version INCLUDEIP - Also include the IP address along with the hostname. Output: <hostname>/<ip address> Example: dns.google/8.8.8.8 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.
<i>hostbyname</i> 3.09	Resolve a Domain Name into an IP Address <pre>▶▶--HOSTBYNAME -----+-----+-----▶▶ +--INCLUDENAME--+ (1)</pre> <ul style="list-style-type: none"> (1) Optional parameter not present in CMS Pipelines Arguments: INCLUDENAME - Also include the name of the host on output. Output: <hostname>/<ip address> Example: dns.google/8.8.8.8
<i>hostid</i> 3.09	Write TCP/IP Default IP Address <pre>▶▶--HOSTID-----+-----+-----▶▶ +--USERid--word- (1) --+</pre> <ul style="list-style-type: none"> (1) The USERid option available under CMS Pipelines is not applicable and is ignored in NetRexx Pipelines
<i>hostname</i> 3.09	Write TCP/IP Host Name <pre>▶▶--HOSTNAME--+-----+--+-----+-----▶▶ +--INCLUDEIP-- (1) -+ +--USERid--word- (2) --+</pre> <ul style="list-style-type: none"> (1) Optional parameter not present in VM/CMS version (2) The USERid option available under CMS is not applicable and is ignored in NetRexx Pipelines Arguments: INCLUDEIP - include the IP address of the system in the response in the form <hostname>/<ip address>

<p><i>htmlrows</i> <i>htmlrow</i> 3.11</p>	<p>Convert rows to HTML format</p> <div data-bbox="309 113 1098 306" style="border: 1px solid black; padding: 5px;"> <p>NetRexx</p> <pre> +-SEPARATOR ", "-----+ ▶--HTMLROWS--++-----+-----+-----+-----+-----▶ +-ROW-QString+ +-SEPARATOR-QString+ ▶-----+-----+-----+-----+-----▶ +-HEAD-QString+ +-DATA-QString+</pre> </div> <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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 one character. Succeeding rows have the standard <td>-tags.
<p><i>httpsplit</i></p>	<p>Split HTTP Data Stream</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>iebcopy</i></p>	<p>Process IEBCOPY Data Format</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>if</i></p>	<p>Process Records Conditionally</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>immcmd</i></p>	<p>Write the Argument String from Immediate Commands</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>insert</i></p>	<p>Insert String in Records</p> <div data-bbox="309 1266 1212 1371" style="border: 1px solid black; padding: 5px;"> <pre> +-BEFORE--+ ▶--INSERT--delimitedString--++-----+-----+-----+-----▶ +-AFTER--+ +-inputRange--+</pre> </div> <ul style="list-style-type: none"> insert a string into a record before or after the record content. Will be much more efficient than specs especially if the string is a Byte[]
<p><i>inside</i></p>	<p>Select Records between Labels</p> <div data-bbox="309 1528 1311 1713" style="border: 1px solid black; padding: 5px;"> <pre> ▶--INSIDE--++-----+-----+-----+-----+-----▶ +-ANYcase-----+ +-delimitedString+ +-CASEANY-----+ +-CASEIGNORE--++ +-IGNORECASE--++ +-CASELESS-----+</pre> </div>
<p><i>instore</i></p>	<p>Load the File into a storage Buffer</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>ip2socka</i></p>	<p>Build sockaddr_in Structure</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.

<i>ispf</i>	Access ISPF Tables <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>jeremy</i>	Write Pipeline Status to the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>join</i>	Join Records <pre> +-1-----+ ▶--JOIN-+-----+-----+-----+-----▶ +-COUNT-+ +-number-----+ +-*-----+ +-KEYLENgth--number-+ ▶+-----+-----+-----+-----+-----▶ +-delimitedString-+-----+ +number-+ +-TERMinate-+ </pre>
<i>joincont</i>	Join Continuation Lines <pre> +-TRAILING-----+ ▶--JOINCONT--+-----+-----+-----+-----▶ +-ANYCase---+ +-NOT-+ +-RANGE--inputRange-+ +-DELAY-+ +-CASEANY---+ +-LEADING-----+ +-CASEIGNORE-+ +-IGNORECASE-+ +-CASELESS---+ ▶+-----+delimitedString-+-----+-----+-----▶ +-ANYof-+ +-KEEP-+ +-delimitedString-+ </pre>
<i>juxtapose</i>	Preface Record with Marker <pre> ▶--JUXTAPOSEe-----+-----+-----▶ +-COUNT-+ </pre>
<i>ldrtbls</i>	Resolve a Name from the CMS Loader Tables <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listcat</i>	Obtain Data Set Names <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listdsi</i>	Obtain Information about Data Sets <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listispf</i>	Read Directory of a Partitioned Data Set into the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listpds</i>	Read Directory of a Partitioned Data Set into the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listzip</i>	List the Files in a Zipped File <pre> NetRexx ▶---LISTZIP---zipFileName-----▶ </pre>
<i>literal</i>	Write the Argument String <pre> ▶---LITERAL---+-----+-----▶ +-string-+ </pre>

Locate

Select Lines that Contain a String

```

▶--LOCATE-+-----+-----+-----+-----+-----+-----+-----▶
      +-ANYCase-----+ +-MIXED-(1)-+ +-inputRanges-+ +-ANYof-+
      +-CASEANY-----+ +-ONEs--(1)-+
      +-CASEIGNORE--+ +-ZEROs-(1)-+
      +-IGNORECASE--+
      +-CASELESS-----+

▶+-----+-----+-----▶
      +-delimitedString-+

```

(1) Not in NetRexx Pipelines, yet.

Lookup

Find Records in a Reference Using a Key Field

```

NetRexx
▶--LOOKUP-+-----+-----+-----+-----+-----+-----+-----▶
      +-COUNT-+ +-ANYCASE-+ +-AUTOADD-+ +-BEFORE-+

▶+-----+-----+-----+-----+-----+-----+-----▶
      +-KEYONLY-+ +-SETCOUNT-+ +-INCREMENT-+ +-TRACKCOUNT-+

▶-----+-----+-----+-----+-----+-----+-----▶
      +-inputRange-+-----+
      +-inputRange-+

▶+-----+-----+-----+-----+-----+-----+-----▶
      +-DETAIL MASTER-----+
      +-DETAIL ALLMASTER PAIRWISE-+
      +-DETAIL ALLMASTER-----+
      +-DETAIL-----+
      +-MASTER DETAIL-----+
      +-MASTER-----+
      +-ALLMASTER DETAIL PAIRWISE-+
      +-ALLMASTER DETAIL-----+
      +-ALLMASTER-----+

```

- 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 propropagate EOFs from output streams.

Find Records in a Reference Using a Key Field

```

CMS
▶▶--LOOKUP--++-----++-----++-----++-----++-----▶▶
                +-COUNT-+ +-MAXcount-number-+ +-INCREMENT-+

                                +-NOPAD-----+
▶+-----++-----++-----++-----++-----▶
                +-SETCOUNT-+ +-TRACKCOUn-+ +-PAD-xorc-+ +-ANYcase-+

▶+-----++-----++-----++-----▶
                +-AUTOADD-+ +-KEYONLY-+ +-STRICT-+
                |           +-BEFORE-+ |
                +-CEILING-----+
                +-FLOOR-----+

▶+-----++-----▶
                +-inputRange-+-----++-+
                +-inputRange-+

                +-DETAIL MASTER-----+
▶+-----++-----▶▶
                +-DETAIL ALLMASTER PAIRWISE-+
                +-DETAIL ALLMASTER-----+
                +-DETAIL-----+
                +-MASTER DETAIL-----+
                +-MASTER-----+
                +-ALLMASTER DETAIL PAIRWISE-+
                +-ALLMASTER DETAIL-----+
                +-ALLMASTER-----+
    
```

- 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 propropagate EOFs from output streams.

Generate a Macro Library from Stacked Members in a COPY File

- Not implemented in Netrexx Pipelines.

<i>mapdisk</i>	Map Minidisks Into Data spaces <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mctoasa</i>	Convert CCW Operation Codes to ASA Carriage Control <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdiskblk</i>	Read or Write Minidisk Blocks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskrandom</i>	Random Access a CMS File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskslow</i>	Read, Append to, or Create a CMS File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskupdate</i>	Replace Records in a File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>members</i>	Extract Members from a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>merge</i>	Merge Streams <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mqsc</i>	Issue Commands to a WebSphere MQ Queue Manager <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>nfind</i> <i>notfind</i>	Select Lines by XEDIT NFind Logic <pre> ▶---+-NFIND---+--+-----+-----▶ +-NOTFIND-+ +--string--+ </pre>
<i>ninside</i> <i>notinside</i> 3.09	Select Records Not between Labels <pre> ▶--+NINSIDE-----+--+-----+delimitedString+-number-----+▶ +-NOTINSIDE--+ +-ANYcase----+ +-delimitedString-+ +-CASEANY-----+ +-IGNORECASE--+ +-CASEIGNORE--+ +-CASELESS----+ </pre>
<i>nlocate</i> <i>notlocate</i>	Select Lines that Do Not Contain a String <pre> ▶--+NLOCATE---+--+-----+--+-----+--+-----+--+-----▶ +-NOTLOCATE-+ +-ANYCase-----+ +-MIXED- (1)-+ +-inputRanges-+ +-CASEANY-----+ +-ONES-- (1)-+ +-CASEIGNORE--+ +-ZEROS- (1)-+ +-IGNORECASE--+ +-CASELESS----+ ▶+-----+--+-----+--+▶ +-ANYof-+ +-delimitedString-+ </pre> <ul style="list-style-type: none"> • (1) Not in NetRexx Pipelines, yet.
<i>noEofBack</i>	Pass Records and Ignore End-of-file on Output <pre> ▶---NOEOFBACK-----▶ </pre>

<i>nop</i>	<p>No Operation</p> <p>NetRexx</p> <pre>▶▶--NOP-----▶▶</pre> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>not</i>	<p>Run Stage with Output Streams Inverted</p> <pre>▶▶--NOT--stage--+-----+-----▶▶ +--operands--+</pre>
<i>notfind</i> <i>nfind</i>	<p>Select Lines by XEDIT NFind Logic</p> <pre>▶▶---+-NOTFIND-+-+-----+-----▶▶ +-NFIND---+ +--string--+</pre>
<i>notin</i> <i>ninside</i>	<p>Select Records Not between Labels</p> <pre>▶▶+-NOTINSIDE--+-----+-----+-----▶▶ +-NINSIDE----+ +-ANYcase----+ +-delimitedString+-number-----+ +-CASEANY----+ +-delimitedString-+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS----+</pre>
<i>notlocate</i> <i>nlocate</i>	<p>Select Lines that Do Not Contain a String</p> <pre>▶▶+-NOTLOCATE-+-+-----+-----+-----+-----▶▶ +-NLOCATE---+ +-ANYCase-----+ +-MIXED- (1)-+ +-inputRanges-+ +-CASEANY-----+ +-ONEs-- (1)-+ +-CASEIGNORE--+ +-ZEROs- (1)-+ +-IGNORECASE--+ +-CASELESS-----+</pre> <pre>▶+-----+-----+-----▶ +-ANYof-+ +-delimitedString-+</pre> <ul style="list-style-type: none"> • (1) Not in NetRexx Pipelines, yet.
<i>nuext</i>	<p>Call a Nucleus Extension</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>optcdj</i>	<p>Generate Table Reference Character (TRC)</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>outside</i>	<p>Select Records Not between Labels</p> <pre>▶▶--OUTSIDE-+-----+-----+-----▶▶ +-ANYcase----+ +-delimitedString-+ +-CASEANY----+ +-delimitedString-+ +-CASEIGNORE-+ +-IGNORECASE-+ +-CASELESS----+</pre>
<i>outstore</i>	<p>Unload a File from a storage Buffer</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>over</i>	<p>Write the Values of Stems</p> <ul style="list-style-type: none"> • Obsolete. Now use varover. over is now an alias for overlay..

<p><i>overlay</i> <i>overla</i> <i>overl</i> <i>over</i></p>	<p>Overlay Data from Input Streams</p> <p>NetRexx</p> <pre> +-NOHOLD- (1) -+ +-PAD- (1)+ +-BLANK-----+ ▶--OVERlay-----+-----+-----+-----▶ +-HOLD- (1) ----+ +-xorC-----+ +-SPACE- (1)+ ▶-----+-----+-----+-----▶ +-TRANSPARENT+-xorC--+(1)+ +-STRING--delimitedString- (1) (2)+ +-BLANK-+ +-SPACE-+ </pre> <ul style="list-style-type: none"> • HOLD keeps the last record from each stream, except primary, and uses it if the stream ends. • TRANSPARENT means that character can be different from the PAD character. If omitted, it is the same as PAD character. • dstream can be used instead of a non-primary stream. • (1) NetRexx Pipelines only • (2) same as highest (+1) stream; implies HOLD
<p><i>overlay</i></p>	<p>Overlay Data from Input Streams</p> <p>CMS</p> <pre> +-BLANK-+ ▶--OVERlay+-----+-----▶ +-xorC--+ </pre>
<p><i>overstr</i></p>	<p>Process Overstruck Lines</p> <ul style="list-style-type: none"> • Not implemented in NetrexX Pipelines.
<p><i>pack</i></p>	<p>Pack Records as Done by XEDIT and COPYFILE</p> <ul style="list-style-type: none"> • Not implemented in NetrexX Pipelines.
<p><i>pad</i></p>	<p>Expand Short Records</p> <pre> +-Right-+ +-BLANK-+ ▶--PAD-+-----+-----+-----+-----number--+-----+-----▶ +-Left--+ +-----+-----MODULO+ +-xorC--+ +-number-+ </pre>
<p><i>parcel</i></p>	<p>Parcel Input Stream Into Records</p> <ul style="list-style-type: none"> • Not implemented in NetrexX Pipelines.
<p><i>parse</i> <i>3.09</i></p>	<p>Rearrange Contents of Records</p> <p>NetRexx</p> <pre> ▶--PARSE--parse_template_Dstring---output_template_Dstring--▶ </pre> <ul style="list-style-type: none"> • Records are parsed via the parse_template_delimited_string. • Variables are named \$n, where n is 1 to 9. • 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. • NetRexX Pipelines only. • . • Example: <pre> parse / 2 \$1 +1/ /The second letter is "\$1". \$\$1 won't be changed./ </pre>
<p><i>pause</i></p>	<p>Signal a Pause Event</p> <ul style="list-style-type: none"> • Not implemented in NetrexX Pipelines.
<p><i>pdsdirect</i></p>	<p>Write Directory Information from a CMS Simulated Partitioned Data Set</p> <ul style="list-style-type: none"> • Not implemented in NetrexX Pipelines.

pick

Select Lines that Satisfy a Relation

NetRexx

```
+ -NOPAD-----+
▶--PICK-+-----+ +-----+
      +-PAD-xorc-+ +-ANYcase- (1) +
                    +-CASEANY-----+
                    +-CASEIGNORE-+
                    +-CASELESS----+
                    +-IGNORECASE-+

▶+-----+ +==-+ +-----+▶
  +-inputRange-+ +-^==-+ +-inputRange-----+
                    +-<<-+ +-delimitedString-+
                    +-<<=-+
                    +->>-+
                    +->>=-+
                    +-\==-+ (2)
                    +-/==-+
                    +-=-+
                    +-^=-+
                    +-<-+
                    +-<=-+
                    +->-+
                    +->=-+
                    +-\==-+ (2)
                    +-/=-+
```

- (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-----+
▶--PICK-----+-----+-----▶
+-PAD--xorc-+ +-ANYcase-----+
+-CASEANY-----+
+-CASEIGNORE--+
+-CASELESS-----+
+-IGNORECASE--+

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

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

List:
|--+-----+-----+-----+-----| Test |--|
+-| List |--+AND--+
+-OR--+

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

CommaList:
-----+
|-----| RangeString |--+---|

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

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

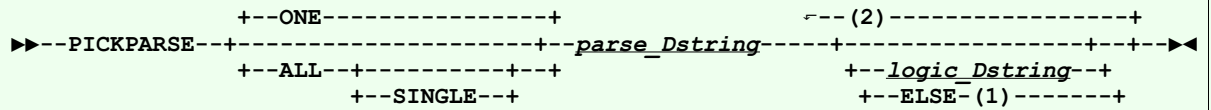
Numeric Operators:
= ^= < <= > >=

```

pickparse
3.09

Select Lines that Satisfy Relations using Rexx Parse

NetRexx



- 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 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 PAF 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 digit. Then those variables can be used in the logic expression. When run, and records matching the logic expression go 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 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 Netrex Pipelines.

pipestop

Terminate Stages Waiting for an External Event

- Not implemented in Netrex Pipelines.

polish

Reverse Polish Expression Parser

- Not implemented in Netrex Pipelines.

predselect

Control Destructive Test of Records

- Not implemented in Netrex Pipelines.

preface

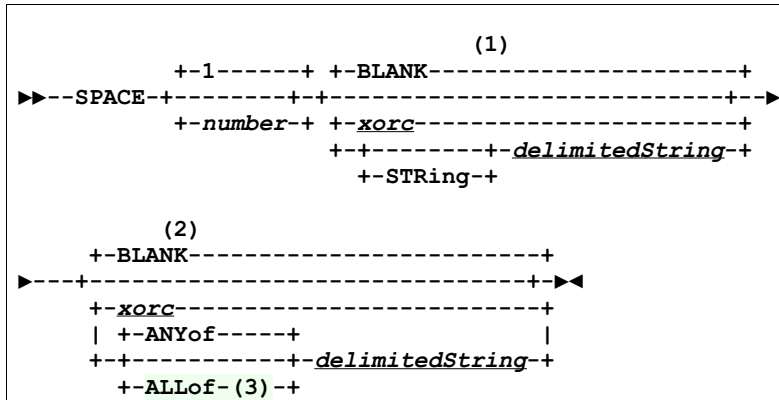
Put Output from a Device Driver before Data on the Primary Input Stream

- Not implemented in Netrex Pipelines.

<i>prefix</i>	<p>Stop and Run a Stage First, Before Continuing</p> <p>NetRexx</p> <pre>▶▶--PREFIX---string-----▶▶</pre> <ul style="list-style-type: none"> • Blocks its primary input and excutes stage supplied as an argument. The output from this stage are put to the primary stream. When its compete the primary input is shorted. • Not implemented in CMS Pipelines.
<i>printmc</i>	<p>Print Lines</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>punch</i>	<p>Punch Cards</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qpdecode</i>	<p>Decode to Quoted-printable Format</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qpencode</i>	<p>Encode to Quoted-printable Format</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qsam</i>	<p>Read or Write Physical Sequential Data Set through a DCB</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>query</i>	<p>Obtain Information From Pipelines</p> <pre> +-VERSION-----+ ▶▶--Query--+-----+▶▶ +-LEVEL-----+ +-SOURCE- (1) ---+ +-MSGLEVEL- (2) -+ +-MSGLIST- (2) --+ </pre> <ul style="list-style-type: none"> • (1) Not CMS • (2) Not NetRexx Pipelines
<i>random</i> 3.09	<p>Generate Pseudorandom Numbers</p> <pre> ▶▶--RANDOM--+-----+▶▶ +-*-----+ +-*-----+ +-+--max_number-----+ +-seed_snumber-+ </pre> <ul style="list-style-type: none"> • NetRexx Pipelines will be a different sequence than CMS gives with the same seed.
<i>reader</i>	<p>Read from a Virtual Card Reader</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>readpds</i>	<p>Read Members from a Partitioned Data Set</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

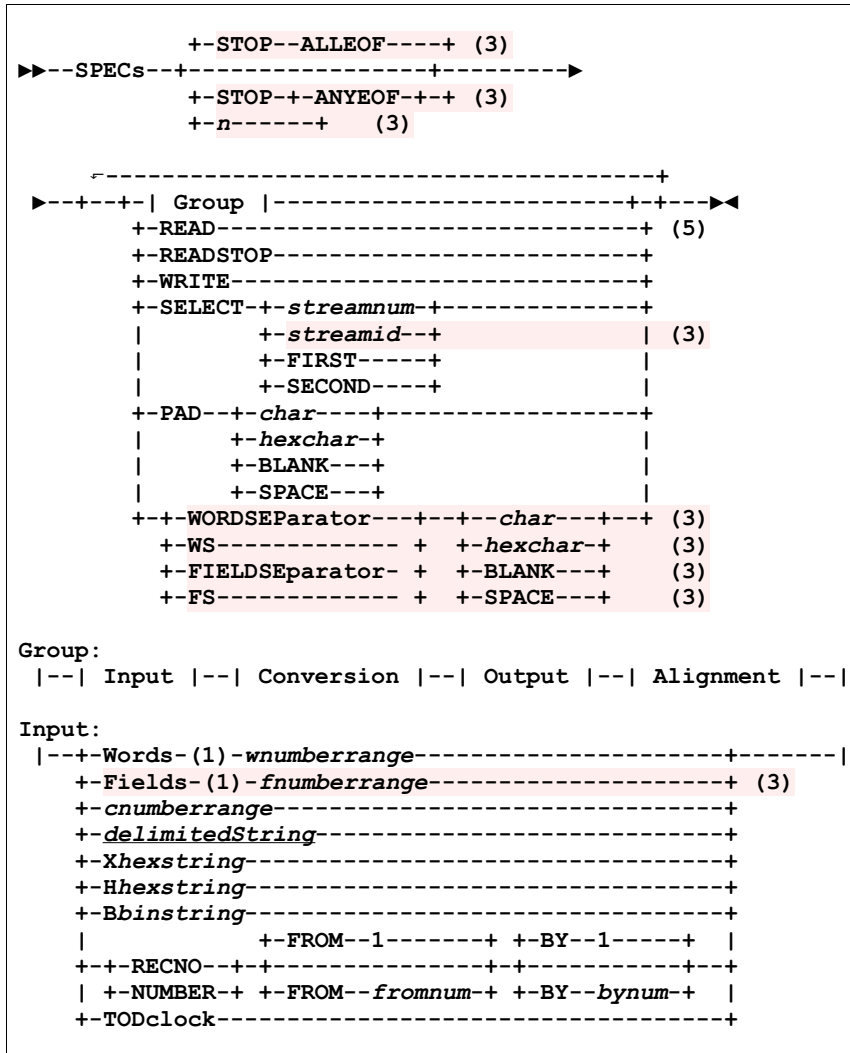
<pre>regex grep 3.09</pre>	<p>Select Lines by a Regular Expression</p> <pre>NetRexx ▶▶---REGEX---▶▶---regex_Dstring- (1)---▶▶ +---GREP---▶▶ +---(-- options_string --)-▶▶ options_string: +-----+ ---+-----+--- +-Numbers-----+ (2) +-Before+-1-----+ (3) +-number-+ +-After+-1-----+ (3) +-number-+ +-Context+-1-----+ (4) +-number-+ +-NOseparator-----+ (5) +-Separator+---/--/--+ (5) +-delimitedString-+ +-Tertiary-----+ (6) +-COUnT-----+ (7)</pre> <ul style="list-style-type: none"> • 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
<pre>retab</pre>	<p>Replace Runs of Blanks with Tabulate Characters</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>reverse</pre>	<p>Reverse Contents of Records</p> <pre>▶▶---REVERSE---▶▶</pre>
<pre>rexx</pre>	<p>Run a REXX Program to Process Data</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>rexxvars</pre>	<p>Retrieve Variables from a REXX or CLIST Variable Pool</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>runpipe</pre>	<p>Issue Pipelines, Intercepting Messages</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>scm</pre>	<p>Align REXX Comments</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>sec2greg</pre>	<p>Convert Seconds Since Epoch to Gregorian Timestamp</p> <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<pre>serialize</pre>	<p>Convert Objects to a Single Text String</p> <pre>NetRexx limited information available</pre> <ul style="list-style-type: none"> • {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

Space Words Like REXX



- (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

Rearrange Contents of Records



Conversion:

```

|---+-----+-----+-----+-----|
+-STRIP-+ +-B2C-----+
+-B2D-----+ (4)
+-B2X-----+ (4)
+-C2B-----+
+-C2D-----+
+-C2F-----+ (3)
+-C2I-----+ (3)
+-C2P-+-----+ (3)
| +- (2) (scale)-+ | (3)
+-C2V-----+ (3)
+-C2X-----+
+-D2C-----+
+-D2X-----+ (4)
+-F2C-----+ (3)
+-I2C-----+ (3)
+-P2C-+-----+ (3)
| +- (2) (scale)-+ | (3)
+-V2C-----+ (3)
+-X2B-----+ (4)
+-X2C-----+
+-X2D-----+ (4)
+-f2t-----+
+-LOWER-----+ (4)
+-UPPER-----+ (4)
+-STRING-----+ (4)

```

Output:

```

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

```

Alignment:

```

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

```

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

```

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

```

- (1) Blanks are optional in this position.
- (2) Blanks are not allowed here.
- (3) CMS only. Not yet implemented in NetRexx Pipelines
- (4) NetRexx Pipelines only. Not yet implemented in CMS
- (5) READ is giving the same output as READSTOP when the streams are different length.
- [6] This senses if it is the first stage, but comment stages will fool it into not producing any output.

spill

Spill Long Lines at Word Boundaries

- Not implemented in Netrex Pipelines.

Split Records Relative to a Target

```
▶--SPLIT--+-----+-----+-----▶
      +-ANYcase-----+ +--MINimum--number--+
      +-CASEANY-----+
      +-CASEIGNORE--+
      +-IGNORECASE--+
      +-CASELESS-----+

      +-AT-----+                +-BLANK-----+
▶--+-----+-----+-----+-----▶
      +-+-----+--+BEFORE+ +NOT+ +-| target |-----+
      +-snumber+ +-AFTER--+                +-number+

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

Interface to SQL

```

+;-+
▶--SQL--+-+-----+-+-----+-+-----+-+-----▶
      +-(-| options |-)-+ +-sql_statement_string-(3)+

options:
-----+
|-----+-----+-----+-----+-----+
|         +-/sqlselect.properties/-+         |
+-PROPERTIES--filename_Qword-(7)-----+(5)+--+
| +-HEADERS---+                               |
+-+         +- (5) (6) -----+
| +-NOHEADERS--+                             |
+-COUNT2SECondary- (5) (11) -----+
+-URL-Qword- (5) (7) -----+
+-JDBCDRIVER-Qword- (5) (7) -----+
+-DBMS-Qword- (5) (7) (8) -----+
+-DB_NAME-Qword- (5) (7) (8) -----+
+-USER-Qword- (5) (7) (8) (10) -----+
+-PASS-Qword- (5) (7) (8) (10) -----+

```

- 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.mssql.MssqlDriver
 - url=jdbc:mssql://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 commandline, 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 d
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. Builtin

sqlcodes

Write the last 11 SQL Codes Received

- Not implemented in Netrexx Pipelines.

```

▶--SQLSELECT--+-----+--▶
      +-(| options |-)-+

+-SELECT--+                               +-;-+
▶+-----+-----+-----+-----+-----▶
      +-sql_select_statement_string-(3)-+

options:
-----+
|-----+-----+-----+-----+-----|
|          +-/sqlselect.properties/-+          |
+-PROPERTIES-+-filename_Qword-(7)-----+(5)+--+
| +-NOHEADERS-+                               |
+-+                               +- (5) (6) -----+
| +-HEADERS-+-----+                         |
+-URL-Qword-(5) (7) -----+
+-JDBCDRIVER-Qword-(5) (7) -----+
+-DBMS-Qword-(5) (7) (8) -----+
+-DB_NAME-Qword-(5) (7) (8) -----+
+-USER-Qword-(5) (7) (8) (10) -----+
+-PASS-Qword-(5) (7) (8) (10) -----+

```

- (1) when using a sqlselect * (all columns) from the commandline, quote the query as in java pipes.compiler (query * from dept | console"
- (2) the netrexx/jdbc combination is extremely case sensitive for column and table names
- (3) if no sql_select_string is specified, it is read from the primary input stream. this is optional in NetRexx Pipelines only. CMS does not use the stream input.
- (4) a maximum of only one record is ever read from the primary input stream.
- (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 SQL stage uses NOHEADERS as the default.
- (10) USER & PASS are needed for some DBMSs and not others, ex. SQLite.
- Priority order for URL, JDBCDRIVER, DBMS, DB_NAME, USER, & PASS (first one 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. Builtin

*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.

*starmsg***Write Lines from a CP System Service**

- Not implemented in Netrexx Pipelines.

*starsys***Write Lines from a Two-way CP System Service**

- Not implemented in Netrexx Pipelines.

<i>state</i>	Provide Information about CMS Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>state</i>	Verify that Data Set Exists <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>statew</i>	Provide Information about Writable CMS Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>stem</i>	Retrieve or Set Variables in a REXX or CLIST Variable Pool <pre> NetRexx ▶--STEM--stem-----▶ </pre>
<i>stem</i>	Retrieve or Set Variables in a REXX or CLIST Variable Pool <pre> CMS ▶--STEM--stem--+-----+-----+-----+-----▶ +-PRODUCER-+ +-number-+ +-NOMSG233-+ +-MAIN-----+ +-SYMBOLIC-+ ▶+-----+-----+-----+-----▶ +-DIRECT---+ +-APPEND-----+ +-FROM--number-+ </pre>
<i>stfle</i>	Store Facilities List <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>storage</i>	Read or Write Virtual Machine Storage <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>strasmfind</i>	Select Statements from an Assembler File as XEDIT Find <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>strasmnfind</i>	Select Statements from an Assembler File as XEDIT NFind <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>strfind</i>	Select Lines by XEDIT Find Logic <pre> ▶-----STRFIND--+-----+--delimitedString-----▶ +-ANYcase-----+ +-CASEANY-----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS----+ </pre> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>strfrlabel</i> <i>strfrlabe</i> <i>strfrlab</i> <i>strfrlab</i>	Select Records from the First One with Leading String <pre> +--STRFROMLABEL--+ +-INCLUSIVE-+ ▶+--STRFRLABEL-+-----+-----+-----+-----▶ +-ANYcase-----+ +-EXCLUSIVE-+ +-CASEANY-----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS----+ </pre>

strfromlabel
strfrlabel
strfrlabe
strfrlab

Select Records from the First One with Leading String

```
      +--STRFROMLABEL--+          +-INCLUSIVE-+
▶---+--STRFRLABEL-----+-----+-----+delimitedString---▶
      +-ANYcase-----+ +-EXCLUSIVE-+
      +-CASEANY-----+
      +-IGNORECASE-+
      +-CASEIGNORE-+
      +-CASELESS---+
```

strip

Remove Leading or Trailing Characters

```
      +-BOTH-----+
▶---STRIP--+-----+-----+-----+-----▶
      +--| case |--+ +-LEADING--+ +-TO--+
      +-TRAILING-+ +-NOT-+

      +-BLANK-----+
▶---+-----+-----+-----+-----▶
      +--| target |--+-----+-----+
      +--number--+ (1)

case:
|-----|
  +--ANYCase-----+
  +--CASEANY-----+
  +--CASEIGNORE--+
  +--IGNORECASE--+
  +--CASELESS-----+

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

- (1) Not implemented in Netrexx Pipelines.

strliteral

Write the Argument String

```
▶---STRLITeral--+-----+-----+-----+-----▶
      | +-PREFACE-+          | +-delimitedString+
      +-----+-----+-----+
      | +-APPEND--+ +-CONDitional-+ |
      +-IFEMPTY-----+-----+
```

strnfind

Select Lines by XEDIT NFind Logic

```
▶---STRNFIND--+-----+-----+-----+-----▶
      +-ANYcase-----+
      +-CASEANY-----+
      +-IGNORECASE-+
      +-CASEIGNORE-+
      +-CASELESS-----+
```


<p><i>strtolabel</i> <i>strtolabe</i> <i>strtolab</i></p>	<p>Select Records to the First One with Leading String</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre style="font-family: monospace;"> +-INCLUSIVE-+ ▶--STRTOlAbel--+-+-----+-----+-----+---delimitedString-----▶ +-ANYcase----+ +-EXCLUSIVE-+ +-CASEANY----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS---+ </pre> </div>
<p><i>structure</i></p>	<p>Manage Structure Definitions</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>strwhilelabel</i> <i>strwhilelabl</i> <i>strwhilelab</i> <i>strwhilela</i> <i>strwhilel</i> <i>strwhile</i> 3.09</p>	<p>Select Run of Records with Leading String</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre style="font-family: monospace;"> +-INCLUSIVE-+ ▶--STRWHILElabel--+-+-----+-----+-----+---delimitedString--▶ +-ANYcase----+ +-EXCLUSIVE-+ +-CASEANY----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS---+ </pre> </div>
<p><i>stsi</i></p>	<p>Store System Information</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>subcom</i></p>	<p>Issue Commands to a Subcommand Environment</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>substring</i></p>	<p>Write substring of record</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>synchronise</i> <i>synchronize</i></p>	<p>Synchronise Records on Multiple Streams</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>synchronize</i> <i>synchronise</i></p>	<p>Synchronise Records on Multiple Streams</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>sysdsn</i></p>	<p>Test whether Data Set Exists</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>sysout</i></p>	<p>Write System Output Data Set</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>sysvar</i></p>	<p>Write System Variables to the Pipeline</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<p><i>tag</i> 3.11</p>	<p>Surrounds Input Records with a HTML tag and its End Tag</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre style="font-family: monospace;"> NetRexx ▶--TAG----string-----▶ </pre> </div> <ul style="list-style-type: none"> • Outputs a record: <tag string>, then passes through all records on its primary input, and finally a record: </tag>. • String may be empty.

<i>trackread</i>	Read Full Tracks from ECKD Device <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>tracksquish</i>	Squish Tracks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>trackverify</i>	Verify Track Format <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>trackwrite</i>	Write Full Tracks to ECKD Device <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>trackxpan</i>	Unsquish Tracks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>translate</i> <i>translat</i> <i>translate</i> <i>transl</i> <i>trans</i> <i>xlate</i>	Transliterate Contents of Records <pre> ▶---+-TRANSlate-+-+-----+-----+-----+-----+-----▶ +--XLATE-----+ +-inputRange-----+ +- default-table --+ +-----+ +--+ (--inputRange--) -+--+ +-----+ ▶---+-+-----+-----+-----+-----+-----▶ +--xrange--xrange--+ default-table: ---+-UPper-----+----- +--LOWer-----+-----+ +--INput-----+-----+ { +--OUTput-----+ } { +---+-TO-----+---+-n---+ } { +--FROM--+ +--CODEPAGE--+ } { } { Not yet in Pipes for NetRexx } </pre>
<i>trfread</i>	Read a Trace File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>truncate</i> <i>truncat</i> <i>trunca</i> <i>trunc</i> <i>chop</i>	Truncate the Record <pre> +--80-----+ ▶---+-TRUNCate-+-+-----+-----+-----+-----+-----▶ +--CHOP-----+ +-snumber----- +--- stringtarget --+ stringtarget: ---+-+-----+-----+-----+-----+----- target -- +-ANYcase-----+ +-BEFORE-+ +-NOT-+ +-CASEANY-----+ +---+-----+-----+-----+ +-CASEIGNORE--+ +-snumber-+ +-AFTER--+ +-IGNORECASE--+ +-CASELESS-----+ target: ---+-xrange-----+----- +---+-STRing---+-delimitedString-+ +--ANYof--+ </pre>

<i>tso</i>	Issue TSO Commands, Write Response to Pipeline <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>udp</i>	Read and Write an UDP Port <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>unique</i> <i>uniqu</i> <i>uniq</i>	Discard or Retain Duplicate Lines <pre> +--NOPAD-----+ ▶--UNIQUE--+-+-----+-----+-----+-----+-----▶ +--COUNT--+ +--PAD--<u>xorc</u>--+ +--ANYcase-----+ +--CASEANY-----+ +--CASEIGNORE--+ +--IGNORECASE--+ +--CASELESS-----+ +--LAST-----+ ▶-----+-----+-----+-----+-----▶ +-- uniqueRanges -+ +--SINGLEs---+ +--FIRST-----+ +--MULTiple--+ +--PAIRwise--+ uniqueRanges: ---+--<u>inputRange</u>-----+-----+-----+-----+----- -----+-----+-----+-----+----- +-- (-+--<u>inputRange</u>-----+-----+-----+-----)---+ +--NOPAD-----+ +--PAD--<u>xorc</u>--+ </pre>
<i>unpack</i>	Unpack a Packed File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>untab</i>	Replace Tabulate Characters with Blanks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>update</i>	Apply an Update File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>urldeblock</i>	Process Universal Resource Locator <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>uro</i>	Write Unit Record Output <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>utf</i>	Convert between UTF-8, UTF-16, and UTF-32 <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>var</i>	Retrieve or Set a Variable in a REXX or CLIST Variable Pool <pre> ▶--VAR--<u>variable</u>-----▶ </pre> <ul style="list-style-type: none"> • Pipes for NetRexx: this can only read vars
<i>vardrop</i>	Drop Variables in a REXX Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>varfetch</i>	Fetch Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

varload	Set Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
varover 3.09	Write the Values of Stems <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> NetRexx ▶▶--VAROVER--varName---▶◀ </div> <ul style="list-style-type: none"> NetRexx Pipelines only; not CMS Pipelines
varset	Set Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
vchar	Recode Characters to Different Length <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
vector	Read or Write an Array of Vectors <ul style="list-style-type: none"> Pipes for NetRexx only.
vectora	Add to an Array of Vectors <ul style="list-style-type: none"> Pipes for NetRexx only.
vectorr	Read From an Array of Vectors <ul style="list-style-type: none"> Pipes for NetRexx only.
vectorw	Write to an Array of Vectors <ul style="list-style-type: none"> Pipes for NetRexx only.
verify 3.09	Verify that Record Contains only Specified Characters <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> ▶▶--VERIFY--++-----+ +-----+ +-----+ (1) +-ANYCASE----+ +-inputRange-+ +-delimitedString-+ +---▶◀ (1) +-CASEANY----+ +-CASEIGNORE-+ +-IGNORECASE-+ +-CASELESS----+ </div> <ul style="list-style-type: none"> (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 <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
vmcdata	Receive, Reply, or Reject a Send or Send/receive Request <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
vmclient	Send VMCF Requests <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
vmclisten	Listen for VMCF Requests <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
waitdev	Wait for an Interrupt from a Device <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.

<i>warp</i>	Pipeline Wormhole <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>warplist</i>	List Wormholes <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>whilelabel</i> 3.09	Select Run of Records with Leading String <pre> ▶---WHILELABEL-+-----+---▶ +-string--+ </pre>
<i>wildcard</i>	Select Records Matching a Pattern <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>writepds</i>	Store Members into a Partitioned Data Set <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>xab</i>	Read or Write External Attribute Buffers <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>xedit</i>	Read or Write a File in the XEDIT Ring <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>xlate</i> <i>translate</i> <i>translat</i> <i>transla</i> <i>transl</i> <i>trans</i>	Transliterate Contents of Records <pre> ▶---+---XLATE-----+---+-----+---▶ +---TRANSLate--+ +---inputRange-----+ +--- default-table ---+ -----+ +---+ (--inputRange--) +---+ -----+ ▶---+---+-----+---+-----▶ +---xrange--xrange--+ default-table: ---+---UPper-----+----- +---LOWer-----+ +---INput-----+ { +---OUTput-----+ } { +---+---TO-----+---+-----+---n--+ } { +---FROM--+ +---CODEPAGE--+ } { } { Not yet in Pipes for NetRexx } </pre>
<i>xmsg</i>	Issue XEDIT Messages <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>xpndhi</i>	Expand Highlighting to Space between Words <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.

xrange
3.09

Write a Range of Characters

```
▶--XRANGE--+-+-----+-----▶◀  
    +-xrange-----|  
    +-xorc--xorc-+
```

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

zone

Run Selection Stage on Subset of Input Record

```
▶--ZONE--+-+-----+-----▶  
    +-+--WORDSEPARATOR--+-+--char-----+--▶  
    +-WS-----+    +-hexchar--+  
    +-FIELDSEPARATOR--+ +-BLANK-----+  
    +-FS-----+    +-SPACE-----+  
  
▶+--Words----wNumberRange-----▶  
    +-Fields----fNumberRange--+  
    +-cNumberRange-----+  
  
▶+-----+-----+-----+stage+-----▶◀  
    +-CASEI--+    +-REVERSE--+    +-operands--+
```

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
end, 7, 12, 25-27, 29
exit, 7, 12, 26, 29
extends, 7, 12, 26, 29
final, 7, 26
forever, 7, 12, 26, 27, 29
import, 7, 12, 26, 29
loop, 7, 12, 25-27, 29
method, 7, 12, 25, 26, 29
options, 29
over, 26
public, 26, 29
rexx, 7, 29
say, 25
signal, 12
static, 25
to, 25, 29
where, 31

ISBN 978-90-819090-3-7

