Pipelines for NetRexx QuickStart Guide

Ed Tomlinson **Jeff Hennick** **René Jansen**

Version 3.09-GA of September 30, 2020

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

Publication Data

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

Contents

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 the formal definition for the language, documenting its syntax and semantics, and prescribing minimal functionality for language implementors. It is the definitive answer to any question on the language, and as such, is subject to approval of the NetRexx Architecture Review Board on any release of the language (including its NRL).
Pipelines for NetRexx QuickStart Guide	The Data Flow oriented companion to NetRexx, with its z/VM CMS Pipelines compatible syntax, is documented in this manual. It discusses installing and running Pipes for NetRexx, and has ample examples of defining your own stages in NetRexx.

Typographical conventions

In general, the following conventions have been observed in the NetRexx publications:

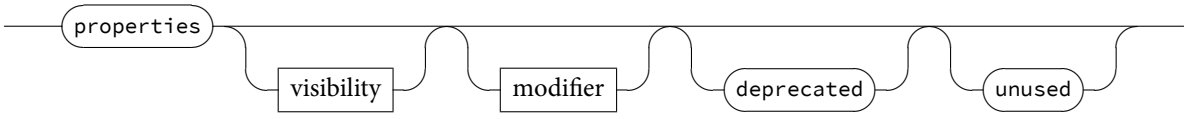
- Body text is in this font
- Examples of language statements are in a **bold** type
- Variables or strings as mentioned in source code, or things that appear on the console, are in a `typewriter` type
- Items that are introduced, or emphasized, are in an *italic* type
- Included program fragments are listed in this fashion:

Listing 1: Example Listing

```
1 -- salute the reader
2 say 'hello reader'
```

- Syntax diagrams take the form of so-called *Railroad Diagrams* to convey structure, mandatory and optional items

Properties



1

Introduction

A Pipeline, or Hartmann Pipeline[1][2], is a concept that extends and improves pipes as they are known from Unix and other operating systems. The name pipe indicates an interprocess communication mechanism, as well as the programming paradigm it has introduced. Compared to Unix pipes, Hartmann Pipelines offer multiple input- and output streams, more complex pipe topologies, and a lot more.

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

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

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

[1] `//https://en.wikipedia.org/wiki/CMS_Pipelines`
[2] This page used to be called Hartmann Pipeline, but was renamed to CMS Pipelines in 2016

2

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 specifiying 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 *producer* and the stage on the right the *consumer*, with the *stage separator* as the connector.

[3] In versions before Pipelines for NetRexx 3.08, the default was the exclamation mark (!), which use was discontinued in favour of conformity with VM/CMS Pipelines.

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 15 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 throug 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.

3

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.\nr{}C"
```

For Windows, the following works: file `pipe.bat`:

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

For Windows, the following works: file `pipc.bat`:

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

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:

```
1 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.

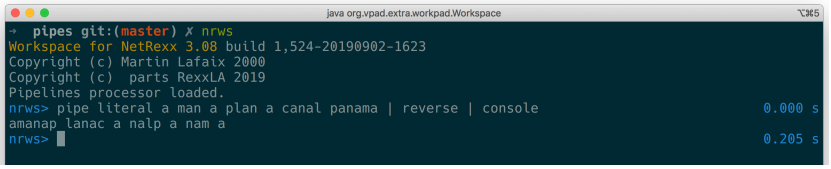


FIGURE 1: Run in the NetRexx Workspace

```
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"
```

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

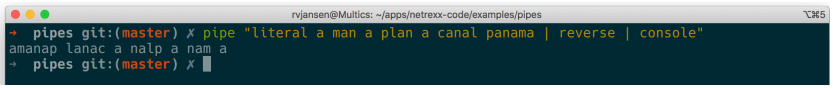


FIGURE 2: Run from the OS command line

3.4 Precompiled Pipelines

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

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

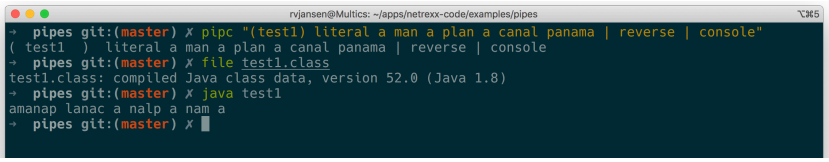


FIGURE 3: Precompile a Pipeline from the OS command line

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 command[4]:

`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:

```
1 pipe (appendtest)
2
3   gen 100 |
4   append gen 50 |
5   rexx locate /0/ |
6   console
```

[4] or an appropriate shortcut in modern shells

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.

4

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 Calender class, run a comparison - get that compiled and test it a bit - an hour or so would be sufficient. Of course, you need to install the Java compiler, because all machines have Java nowadays, but just not the compiler. But if you want to really impress people, you should type in a command line and be done with it. For this you can use NetRexx pipelines. Fortunately, you emailed the NetRexxF.jar to yourself so you save it on the machine, and you're in business right away; you add it to the classpath. Your first pipeline command should just test the waters. For this chapter, we will use the

```
nrws
```

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

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

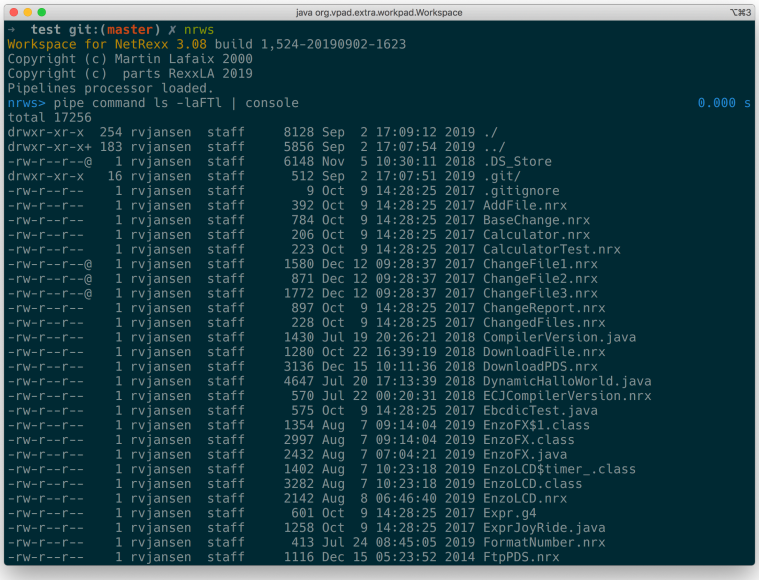


FIGURE 4: example 1

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
```

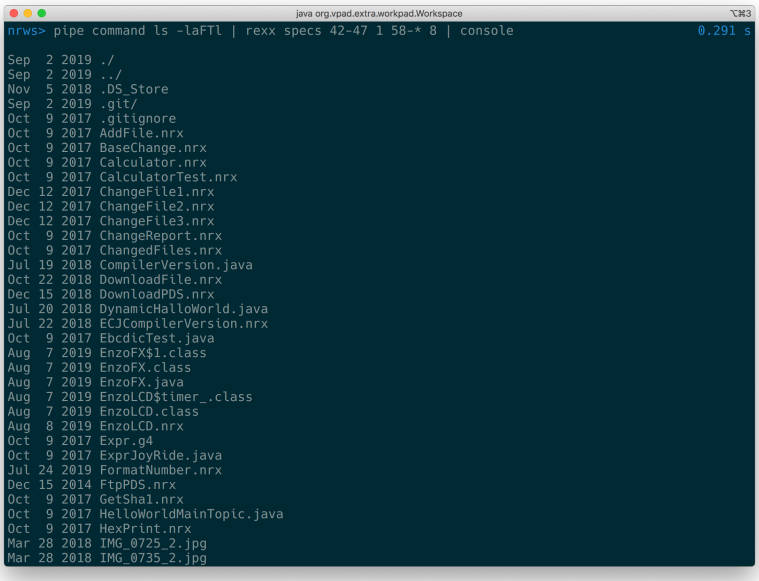


FIGURE 5: example 2

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 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)
```

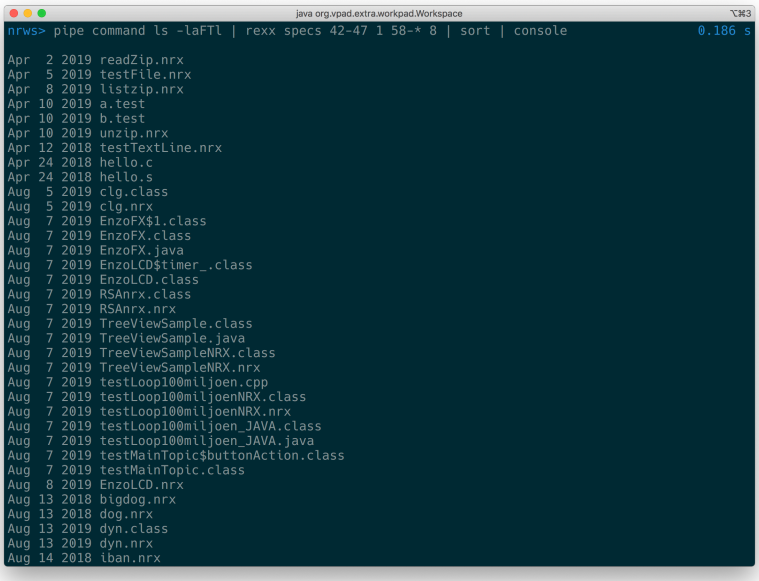


FIGURE 6: example 3

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

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

```
pipc commandtest && java newfiles
```

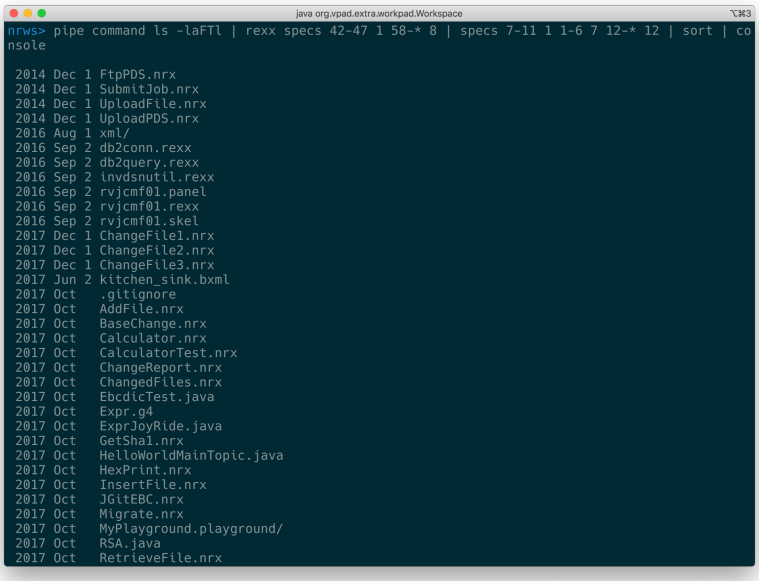


FIGURE 7: example 4

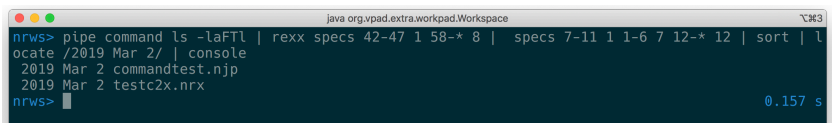


FIGURE 8: example 5

5

Write your own Filters

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

```
 /* 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))
```

So that would look fairly familiar, and admittedly, a bit easier for us already well versed in NetRexx. We can test this by building a pipeline and running the filter on its own source:

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

[5] From the document CMS Pipelines Explained, by John P. Hartmann

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.

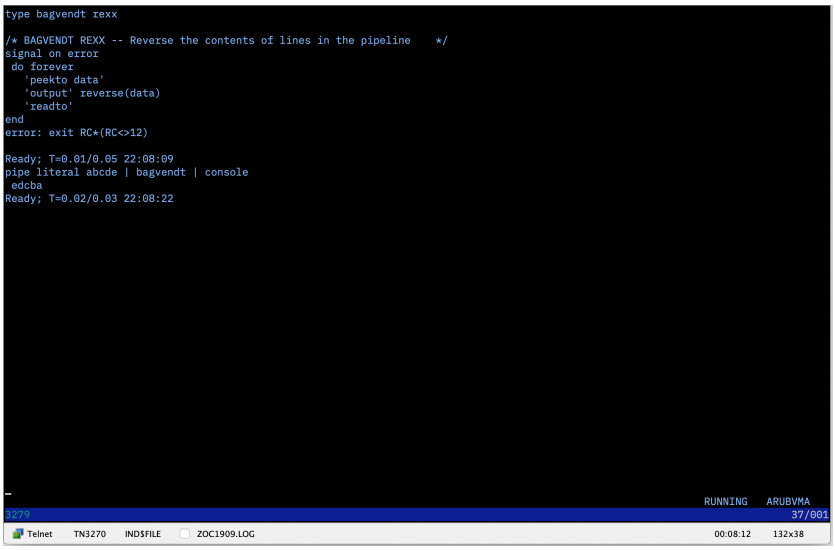


FIGURE 9: BAGVENDT under VM/CMS

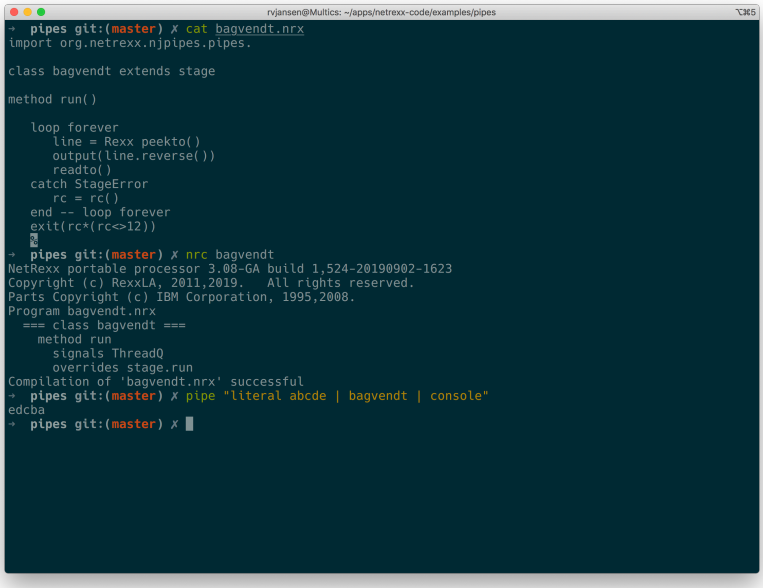


FIGURE 10: bagvendt.nrx under NetRexx

6

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 tast 2...
 b: ?

 a: |
  copy |
  task 3 |
 b:
```

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

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

7

Device Drivers

Pipelines for NetRexx contains the following device drivers:

<	read from a fle
>	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 fle
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

8

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
frlabel	select records from the first one with leading string
strfrlabel	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

9

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

10

Other Stages

query	check version and level of Pipelines for NetRexx
– –	insert comments into a pipeline
comment	insert comments into a pipeline

11

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:

```
1 pipe literal foo bar baz frob frobnitz frobbotzim | split | locate /oo/ |
2 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.

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

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

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

The output is now:

```
bar
baz
frob
frobnitz
frobbotzim
```

[7] often, you will see it being called 'a:'

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.

12

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:

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

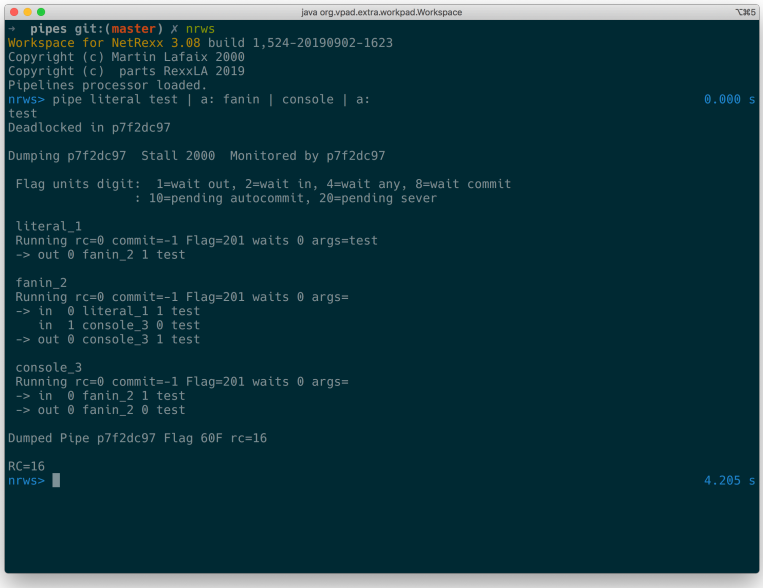


FIGURE 11: Deadlock detection

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 syncronized blocks.

When a stage has a pending sever or autocommit, flag bits are set too.

13

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 similiar effect.

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

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

In .njp files you can use avar phrase actually just shorthand for arg('avar'). The following example shows what has to be done in a stage to access the rexx variables passed by VAR, STEM and OVER. The real over stage is a bit more complete.

```
  -- 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
```

```
4    b = 'some text for literal'
5
6    addpipe (linktest) literal {b} | dup {a} | *in0:
7
8    loop forever
9      line = Rexx readto()
10   catch StageError
11   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.

14

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
```

15

TCP/IP Networking using Pipes for NetRexx

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 oneresponse |
  rexx to console

-- a single tasking server

options binary
import org.netrexx.njpipes.pipes.
class tcpexample extends stage

method run() public

  loop forever

    peekto()

    callpipe (tcplog stall 15000 debug 0)
      *in0: |
      take first 1 |
      console |
    f: fanin |
      tcpdata timeout 10000 deblock C oneresponse |
      elastic |
      insert /\n/ after |
    f:

  catch StageError
    rc = rc()
  end

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.

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

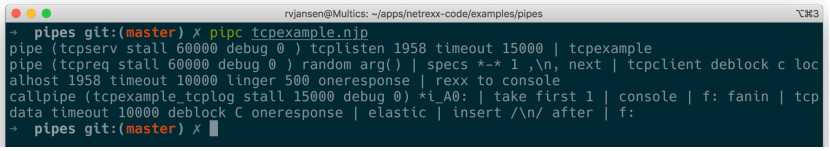


FIGURE 12: TCP/IP Client/Server compile

arbitrary, but specified in the pipeline source.

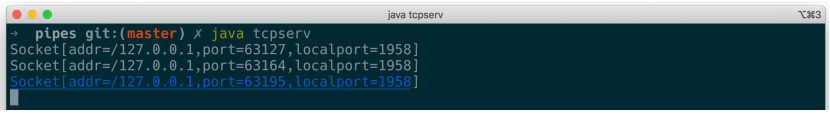


FIGURE 13: TCP/IP server

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.

FIGURE 14: TCP/IP requestor

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

16

Selecting from databases with Pipelines for NetRexx

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.

17

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 persistence[8].

The *pipe* command alias start the Pipes Runner, which is a command processor that can execute a pipe from the command line in an OS shell, the OS being Windows, Linux or macOS[9].

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

```
1 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.

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

18

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.

19

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
```

Stages Built Into NetRexx Pipelines 3.09 & 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 andtop 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.

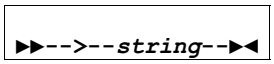
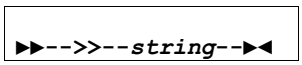
Pipelines Builtin Stages

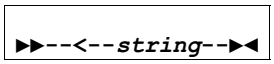
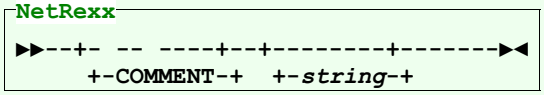
Show Stages Implemented in:

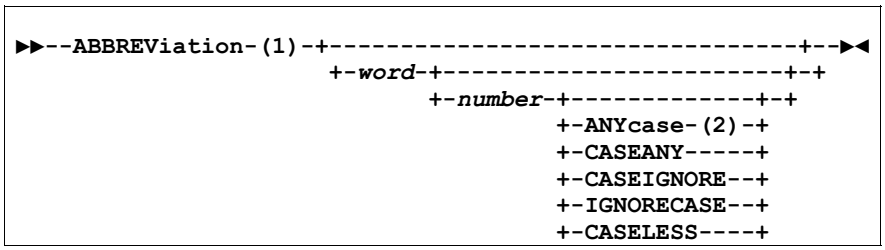
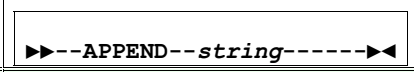
NetRexx Pipelines: ☑ **CMS Pipelines:** ☑

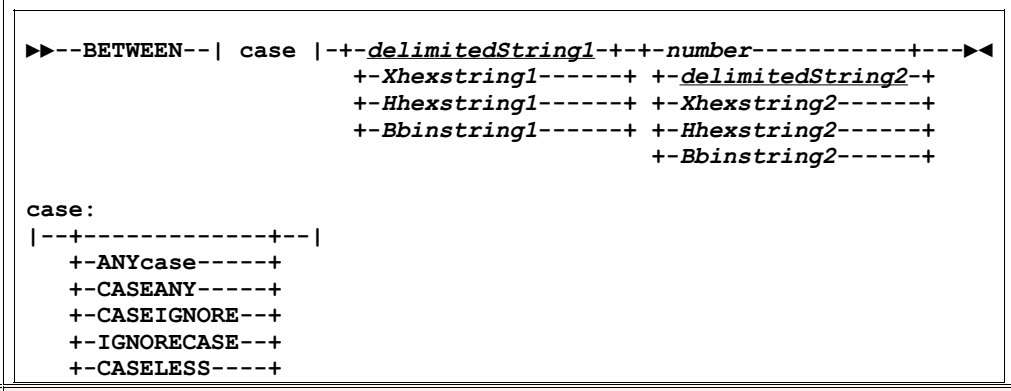
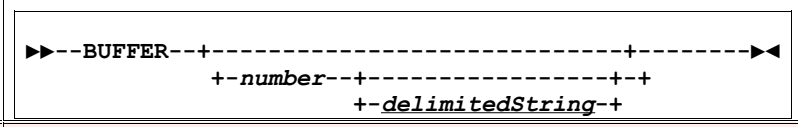
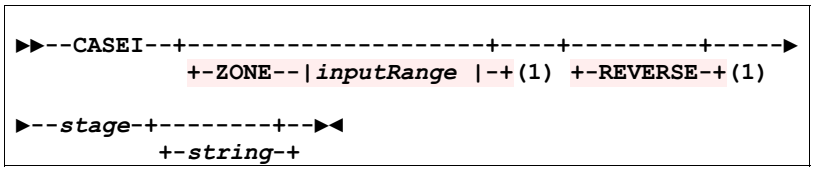
Show All Details: ☑ (Double click on a row to turn it on/off.)

Highlight NetRexx Only / CMS Only: ☑

> ***diskw*** ***filew*** *3.09*	**Replace or Create a File**  • delegates to **diskw**.
>> ***diska*** ***filea*** *3.09*	**Append to or Create a File**  • delegates to **diska**.
>>mdsk	**Append to or Create a CMS File on a Mode** • Not implemented in Netrexx Pipelines.
>>mvs	**Append to a Physical Sequential Data Set** • Not implemented in Netrexx Pipelines.
>>oe	**Append to or Create an OpenExtensions Text File** • Not implemented in Netrexx Pipelines.
>>sfs	**Append to or Create an SFS File** • Not implemented in Netrexx Pipelines.
>>sfsslow	**Append to or Create an SFS File** • Not implemented in Netrexx Pipelines.
>mdsk	**Replace or Create a CMS File on a Mode** • Not implemented in Netrexx Pipelines.

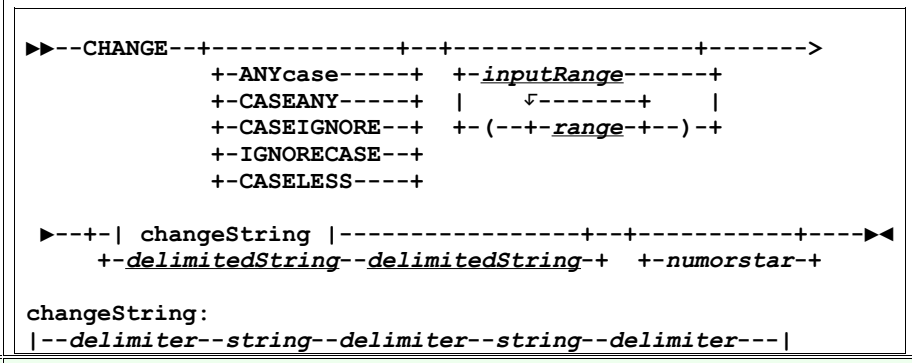
>mvs	**Rewrite a Physical Sequential Data Set or a Member of a Partitioned Data Set** • Not implemented in Netrexx Pipelines.
>oe	**Replace or Create an OpenExtensions Text File** • Not implemented in Netrexx Pipelines.
>sfs	**Replace or Create an SFS File** • Not implemented in Netrexx Pipelines.
< *diskr* *disk* *file* *filer* *3.09*	**Read a File**  • Implemented as in CMS; delegates to **diskr**.
<mdsk	**Read a CMS File from a Mode** • Not implemented in Netrexx Pipelines.
<mys	**Read a Physical Sequential Data Set or a Member of a Partitioned Data Set** • Not implemented in Netrexx Pipelines.
<oe	**Read an OpenExtensions Text File** • Not implemented in Netrexx Pipelines.
<sfs	**Read an SFS File** • Not implemented in Netrexx Pipelines.
<sfsslow	**Read an SFS File** • Not implemented in Netrexx Pipelines.
-- *comment* *3.09*	**Comment Stage, No Operation**  • 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** • Not implemented in Netrexx Pipelines.
3277enc	**Write the 3277 6-bit Encoding Vector** • Not implemented in Netrexx Pipelines.
64decode	**Decode MIME Base-64 Format** • Not implemented in Netrexx Pipelines.
64encode	**Encode to MIME Base-64 Format** • Not implemented in Netrexx Pipelines.
? *help*	**Display Help for Pipelines** • Not implemented in Netrexx Pipelines.

abbreviation ***abbreviatio*** ***abbreviati*** ***abbreviat*** ***abbrevi*** ***abbrev***	**Select Records that Contain an Abbreviation of a Word in the First Positions**  • (1) ABBREViation must be ABBREV in CMS • (2) ANYcase must be ANYCASE in CMS
acigroup	**Write ACI Group for Users** • Not implemented in Netrexx Pipelines.
addrdw	**Prefix Record Descriptor Word to Records** • Not implemented in Netrexx Pipelines.
adrspace	**Manage Address Spaces** • Not implemented in Netrexx Pipelines.
aftfst	**Write Information about Open Files** • Not implemented in Netrexx Pipelines.
aggrc	**Compute Aggregate Return Code** 
all	**Select Lines Containing Strings (or Not)** • Not implemented in Netrexx Pipelines.
alserv	**Manage the Virtual Machine's Access List** • Not implemented in Netrexx Pipelines.
apldecode	**Process Graphic Escape Sequences, Old APL language** • Not implemented in Netrexx Pipelines.
aplencode	**Generate Graphic Escape Sequences, Old APL language** • Not implemented in Netrexx Pipelines.
append	**Put Output from a Device Driver after Data on the Primary Input Stream** 
array	**Read or Write an Array** • Pipes for NetRexx
arraya	**Read or Write an Array** • Pipes for NetRexx
arrayr	**Read or Write an Array** • Pipes for NetRexx
arrayw	**Read or Write an Array** • Pipes for NetRexx

asatomc	**Convert ASA Carriage Control to CCW Operation Codes. Old printer control** • Not implemented in Netrexx Pipelines.
asmcont	**Join Multiline Assembler Statements** • Not implemented in Netrexx Pipelines.
asmfind	**Select Statements from an Assembler File as XEDIT Find** • Not implemented in Netrexx Pipelines.
asmnfind	**Select Statements from an Assembler File as XEDIT NFind** • Not implemented in Netrexx Pipelines.
asmxpnd	**Expand Joined Assembler Statements** • Not implemented in Netrexx Pipelines.
beat	**Mark when Records Do not Arrive within Interval** • Not implemented in Netrexx Pipelines.
between *3.09*	**Select Records Between Labels** 
block	**Block to an External Format** • Not implemented in Netrexx Pipelines.
browse	**Display Data on a 3270 Terminal** • Not implemented in Netrexx Pipelines.
buffer	**Buffer Records** 
c14to38	**Combine Overstruck Characters to Single Code Point. Old printer.**
casei	**Run Selection Stage in Case Insensitive Manner**  • (1) CMS Pipelines only.

change

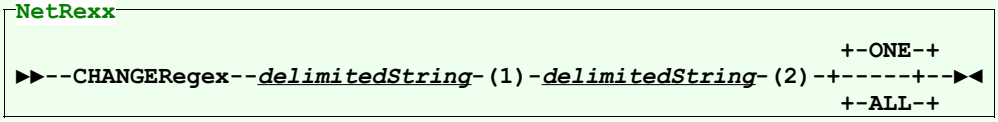
Substitute Contents of Records



changeregex
changerege
changereg
changere
changer
3.09

Substitute Contents of Records using Java Regular Expressions

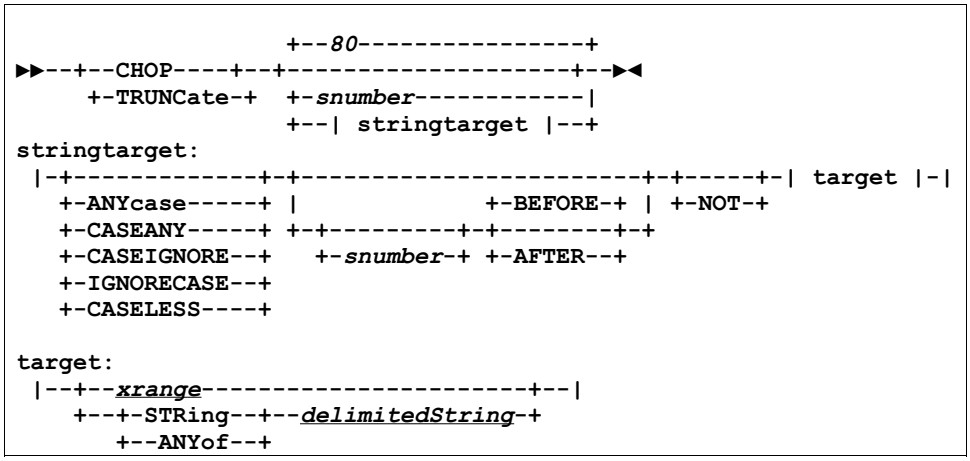
NetRexx



- Uses the Java RegEx classes and its dialect of RegEx. See Java's **Pattern** class and **replaceFirst** and **replaceAll** methods of **String** for full documentation.
- (1) First delimitedString is a Java *RegEx expresion* for what is to be replaced.
- (2) Second delimitedString is the replacement string. It may contain elements from the first one.

chop
truncate
truncat
trunca
trunc

Truncate the Record



cipher

Encrypt and Decrypt Using a Block Cipher

- Not implemented in Netrexx Pipelines.

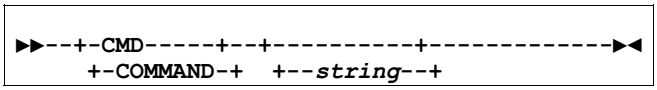
ckddeblock

Deblock Track Data Record

- Not implemented in Netrexx Pipelines.

cmd
command

Issue OS Commands, Write Response to Pipeline



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

cms

Issue CMS Commands, Write Response to Pipeline '

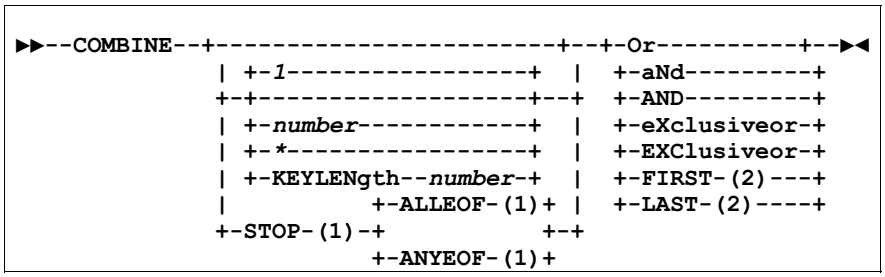
collate

Collate Streams

- Not implemented in Netrexx Pipelines.

combine

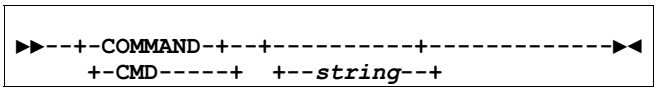
Combine Data from a Run of Records



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

command
cmd

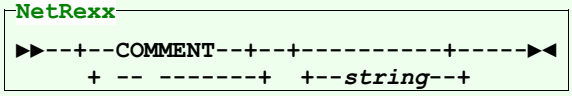
Issue OS Commands, Write Response to Pipeline



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

comment
--
3.09

Comment stage



- Not in CMS Pipelines;
- This is a STAGE, not a programming comment. It must have a SPACE after --.
- It must have either a stageEnd or pipeEnd.
- If ended with a stageEnd, it passes records through on primary input to output streams.
- If ended with a pipeEnd, it does NOT pass records through.
- If used before a driver stage, it must have a pipeEnd.

compare

Compare Primary and Secondary Streams, Write the Result

NetRexx

```
              +-TRINARY-+  (1)                    +-PAD SPACE-+
▶▶--COMPARE--+---------+--------------------+---+-----------+--+------+--▶◀
              +-BINARY--+  (2)                |   +-PAD-xorc--+  +-ECHO-+
              |                               |
              | ↓-------------------------+   |
              +--ANY delimitedString------+--+   (4) (5)
              +--EQUAL delimitedString----+      (4)
              +--LESS delimitedString-----+  (3) (4)
              +--MORE delimitedString-----+  (3) (4)
              +--NOTEQUAL delimitedString-+      (4)
```

- (1) -1 = Primary is shorter/less, 0 = equal, 1 = Secondary is shorter/less
- (2) 0 = equal, 1 = not equal
- (3) Primary is LESS/shorter (or MORE/longer) than secondary
- (4) *DStrings* can use any of the following escapes (or the lowercase) for the unequal situation:
  - \C (count) for the record number,
  - \B (byte) for column number
  - \P (primary) for the primary stream record
  - \S (secondary) for the secondary stream record
  - \L (Least) for the stream number that is shorter, -1 if equal
  - \M (Most) for the stream number that is longer, -1 if equal
- (5) Equal or not, this DString precedes any of the others.
- (6) This is NetRexx Pipelines only, not included in CMS
- (7) In reporting \P & \S, control charactors, 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 non-compare primary input stream records continue to be read and reported, but no testing is done.
- (9) Options work in any order
- Input streams:
  - 0: Data 1
  - 1: Data 2
- Output streams:
  - 0: Result (single record, possibly multiple lines)
  - 1: Last primary record read at first no match, or end of stream
  - 2: Last secondary record read at first no match, or end of stream

configure

Set and Query CMS Pipelines Confguration Variables

- Not implemented in Netrexx Pipelines.

console
consol
conso
cons
cons
terminal
termina
termin
termi
term

Read or Write the Terminal in Line Mode

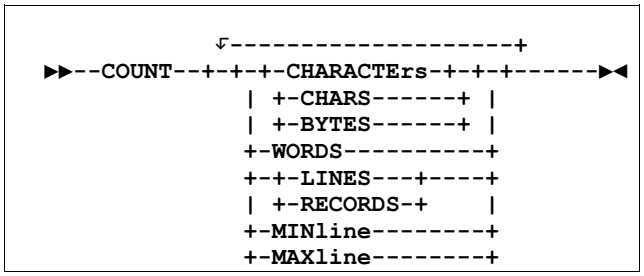
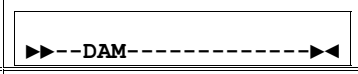
```
▶▶--+-CONSole--+--+----------------------+--+--------------------+--▶◀
    +-TERMinal-+  +-EOF--delimitedString-+  +-DIRECT-(1)---------+
                  +-NOEOF----------------+  +-ASYNchronously-(1)-+
                                            +-DARK-(1)-----------+
```

- (1) CMS Pipelines Only.

copy

Copy Records, Allowing for a One Record Delay

```
▶▶--COPY---------▶◀
```

count	**Count Lines, Blank-delimited Words, and Bytes** 
cp	**Issue CP Commands, Write Response to Pipeline** • Not implemented in Netrexx Pipelines.
crc	**Compute Cyclic Redundancy Code** • Not implemented in Netrexx Pipelines.
dam	**Pass Records Once Primed** 
dateconvert *dateconver* *dateconve* *dateconv* *3.09*	**Convert Date Formats**

```
▶▶--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     }   mm/dd/yy hh:mm:ss.uuuuuu
USA_SHORT     }
REXX_DATE_U   }

FULldate      }   mm/dd/yyyyyyy hh:mm:ss.uuuuuu
USA           }

ISO_SHORT         yy-mm-dd hh:mm:ss.uuuuuu
ISOdate           yyyyyyy-mm-dd hh:mm:ss.uuuuuu
DB2_SHORT         yy-mm-dd-hh.mm.ss.uuuuuu
DB2               yyyyyyy-mm-dd-hh.mm.ss.uuuuuu
VMDATE  (2)
NORMAL            dd mmm yyyyyyy hh:mm:ss.uuuuuu
CSL_SHORT     }   yy/mm/dd hh:mm:ss.uuuuuu
REXX_DATE_O   }
CSL               yyyyyyy/mm/dd hh:mm:ss.uuuuuu
PIPE_SHORT        yymmddhhmmssuuuuuu
PIPE          }   yyyymmddhhmmssuuuuuu
REXX_DATE_S   }
EURSHORT          dd.mm.yy hh:mm:ss.uuuuuu
EUR               dd.mm.yyyyyyy hh:mm:ss.uuuuuu
JULIAN_SHORT      yy.ddd hh:mm:ss.uuuuuu
JULIAN            yyyyyyy.ddd hh:mm:ss.uuuuuu
TOD_ABSOLUTE  }  (2)
TODABS        }  (2)
SCIENTIFIC_ABSOLUTE }  (2)
SCIABS        }  (2)
POSIX             sssssssss
TOD_RELATIVE  }  (2)
TODREL        }  (2)
SCIENTIFIC_RELATIVE }  (2)
SCIREL        }  (2)
MET  (2)
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/yyyyyyy 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 yyyyyyy hh:mm:ss.uuuuuu
REXX_DATE_W       wwwwwwwww (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.
- CMS has many more mainframe centric formats that NetRexx Pipelines does not process.
- (1) Not CMS Pipelines.

deal

Pass Input Records to Output Streams Round Robin

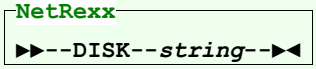
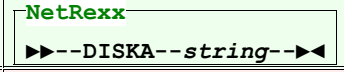
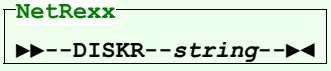
```
          +-STOP--ALLEOF-----------------+
▶▶--DEAL--+-----------------------------+--------▶◀
          +-STOP-+-ALLEOF-(2)+-------------+
          |      +-ANYEOF----+             |
          |      +-number----+             |
          +-SECONDARY---------+--------+
          |                +-RELEASE---+   |
          |                +-LATCH-(1)-+   |
          +-KEY--inputRange--+-------+------+
          |                  +-STRIP-+      |
          +-STREAMid--inputRange--+-------+-+
                                  +-STRIP-+
```

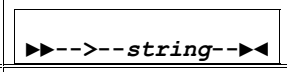
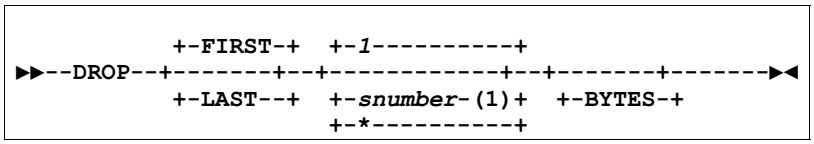
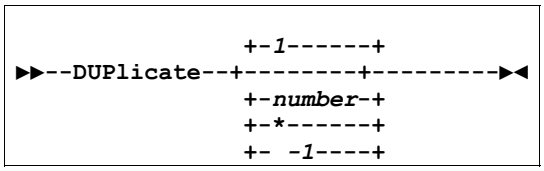
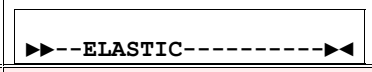
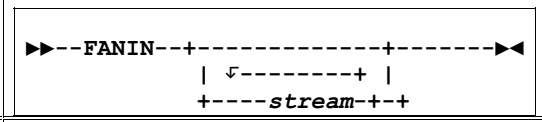
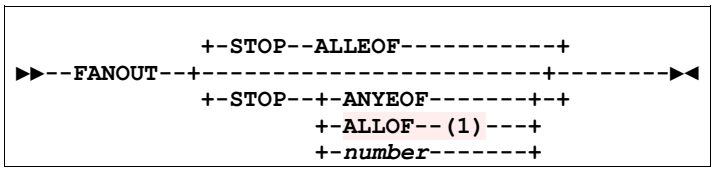
- (1) ...exx Pipelines
- Since NetRexx dispatches the stages threads, DEAL may not see a sever immediately, as the severing thread can get multitasked. This can make options like ANYEOF work in unexpected ways.

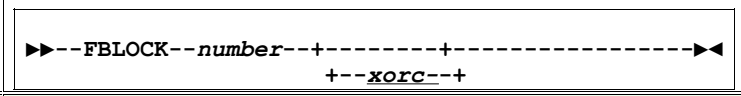
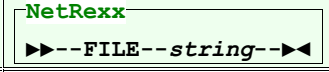
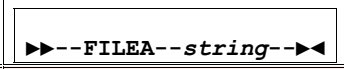
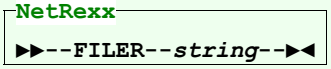
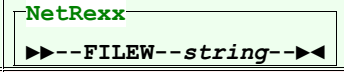
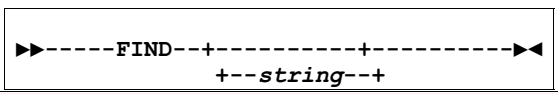
deblock

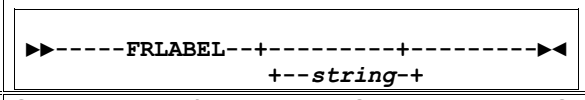
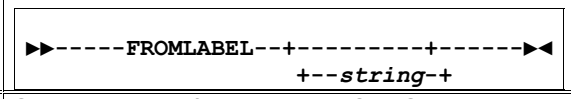
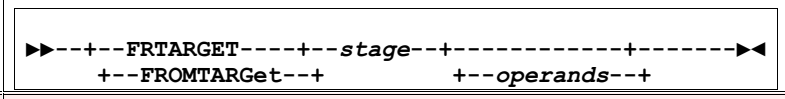
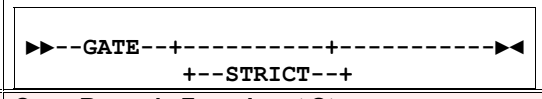
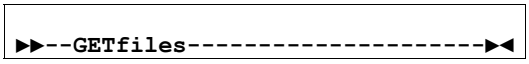
Deblock External Data Formats

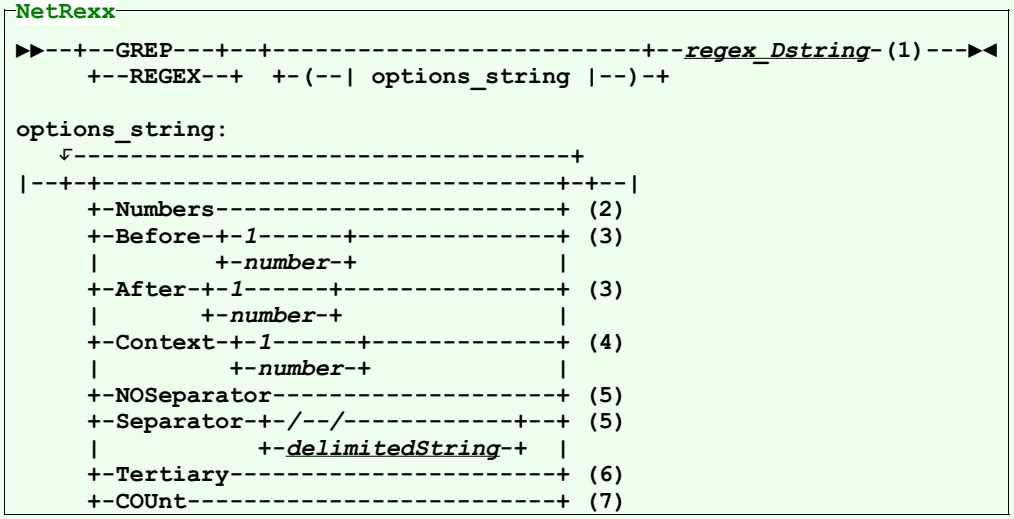
```
                     +-80--+
              +--FIXED--+-number-+-------+------------+
              |                   +-PAD--xorc--+ (1)  |
▶▶--DEBLOCK--+-----------------------------------------------+-▶◀
              +-[illegible]---------+-+-----------+-+-----+
              +-[illegible]                +-TERMINATE- +-EOF-+
              +-[illegible]-+
              +-LINEND--xorc---+
              +-STRING--delimitedString-+
```

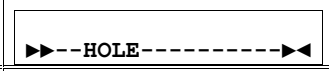
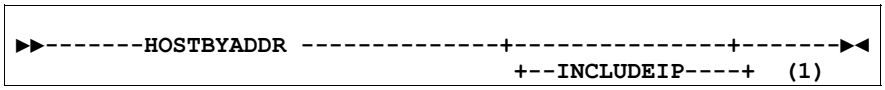
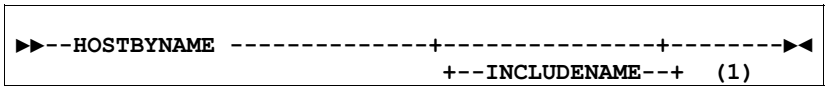
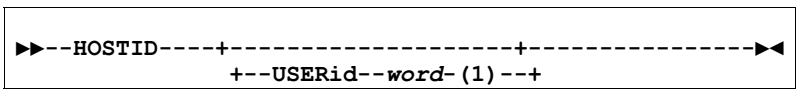
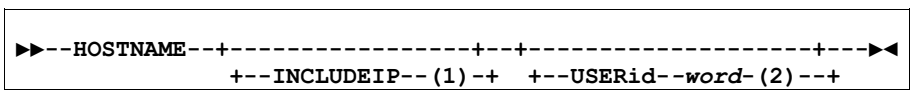
delay	**Suspend Stream** • Not implemented in Netrexx Pipelines.
devinfo	**Write Device Information** • Not implemented in Netrexx Pipelines.
dfsort	**Interface to DFSORT/CMS** • Not implemented in Netrexx Pipelines.
diage4	**Submit Diagnose E4 Requests** • Not implemented in Netrexx Pipelines.
dict ***hash***	**Read or Write a Dictionary** • Pipes for NetRexx only.
dicta ***hasha***	**Read or Write a Dictionary** • Pipes for NetRexx only.
dictr ***hashr***	**Read or Write a Dictionary** • Pipes for NetRexx only.
dictw ***hashw***	**Read or Write a Dictionary** • Pipes for NetRexx only.
digest	**Compute a Message Digest** • Not implemented in Netrexx Pipelines.
disk ***diskr*** **<** ***file*** ***filer***	**Read a File**  • As in CMS, equivalent to diskr (Pipes for NetRexx Only) or **<**.
diska **>>** ***filea***	**Append to or Create a File** 
diskback ***fileback***	**Read a File Backwards** • Not implemented in Netrexx Pipelines.
diskfast ***filefast***	**Read, Create, or Append to a File** • Not implemented in Netrexx Pipelines.
diskid	**Map CMS Reserved Minidisk** • Not implemented in Netrexx Pipelines.
diskr ***disk*** **<** ***file*** ***filer***	**Read a File**  • As in CMS, equivalent to diskr (Pipes for NetRexx Only) or **<**.
diskrandom ***filerandom***	**Random Access a File** • Not implemented in Netrexx Pipelines.

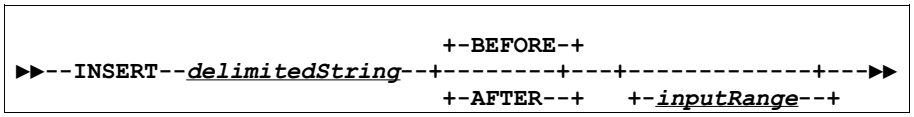
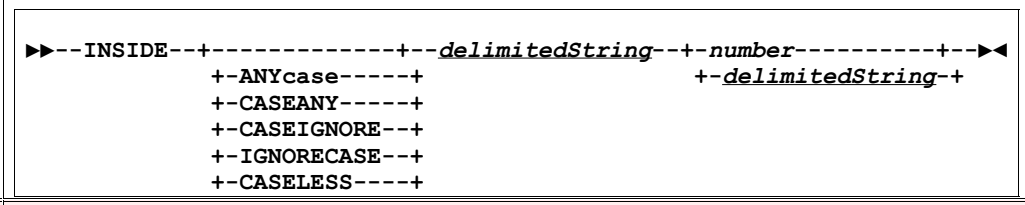
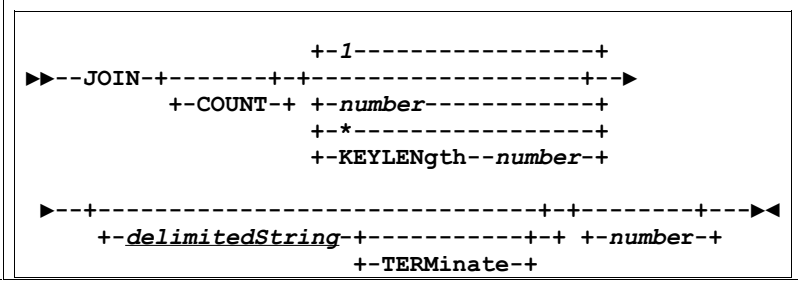
diskslow ***fileslow***	**Read, Create, or Append to a File** • Not implemented in Netrexx Pipelines.
diskupdate ***fileupdate***	**Replace Records in a File** • Not implemented in Netrexx Pipelines.
diskw ***filew*** ***>***	**Replace or Create a File** 
drop	**Discard Records from the Beginning or the End of the File**  • (1) CMS: must be positive. NetRexx Pipelines: negative reverses FIRST/LAST, so DROP FIRST -3 is the same as DROP LAST 3.
duplicate ***duplicat*** ***duplica*** ***duplic*** ***dupli*** ***dupl*** ***dup***	**Copy Records**  • (1) CMS is DUPlicat due to 8-character name limitation
elastic	**Buffer Sufficient Records to Prevent Stall** 
eofback	**Run an Output Device Driver and Propagate End-of-?le Backwards** • Not implemented in Netrexx Pipelines.
escape	**Insert Escape Characters in the Record** • Not implemented in Netrexx Pipelines.
fanin	**Concatenate Streams** 
faninany	**Copy Records from Whichever Input Stream Has One** 
fanintwo	**Pass Records to Primary Output Stream**
fanout	**Copy Records from the Primary Input Stream to All Output Streams**  • (1) CMS only
fanouttwo	**Copy Records from the Primary Input Stream to Both Output Streams**

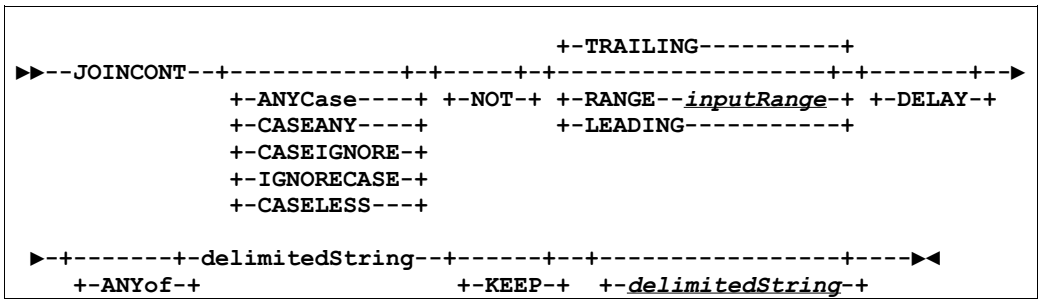
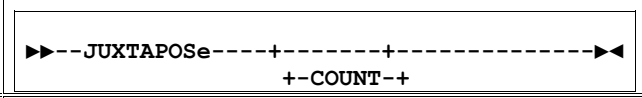
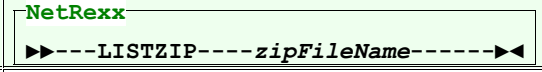
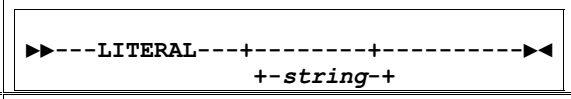
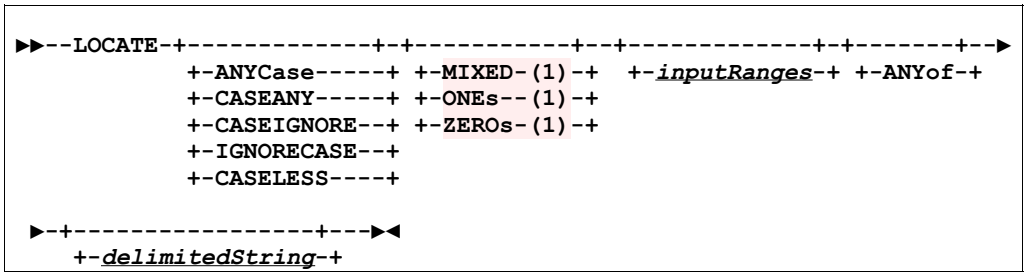
fbaread	**Read Blocks from a Fixed Block Architecture Drive** • Not implemented in Netrexx Pipelines.
fbawrite	**Write Blocks to a Fixed Block Architecture Drive** • Not implemented in Netrexx Pipelines.
fblock	**Block Data, Spanning Input Records** 
file ***filer*** ***disk*** ***diskr***	**Read a File** NetRexx 
filea ***diska*** ***>>***	**Append to or Create a File** 
fileback ***diskback***	**Read a CMS file backwards** • Not implemented in Netrexx Pipelines.
filedescriptor	**Read or Write an OpenExtensions File that Is Already Open** • Not implemented in Netrexx Pipelines.
filefast ***diskfast***	**Read or write a CMS file** • Not implemented in Netrexx Pipelines.
filer ***file*** ***disk*** ***diskr*** ***<***	**Read a File** NetRexx 
filerandom ***diskrandom***	**Read specific records from a CMS file**
fileslow ***diskslow***	**Read, Create, or Append to a File** • Not implemented in Netrexx Pipelines.
filetoken	**Read or Write an SFS File That is Already Open** • Not implemented in Netrexx Pipelines.
fileupdate ***diskupdate***	**Change records in a CMS file**
filew ***diskw*** ***>***	**Replace or Create a File** NetRexx 
fillup	**Pass Records To Output Streams** • Not implemented in Netrexx Pipelines.
filterpack	**Manage Filter Packages** • Not implemented in Netrexx Pipelines.
find	**Select Lines by XEDIT Find Logic** 
fitting	**Source or Sink for Copipe Data** • Not implemented in Netrexx Pipelines.

fmtfst	**Format a File Status Table (FST) Entry** • Not implemented in Netrexx Pipelines.
frlabel ***fromlabel***	**Select Records from the First One with Leading String** 
fromlabel ***frlabel***	**Select Records from the First One with Leading String** 
frtarget	**Select Records from the First One Selected by Argument Stage** 
fullscrq	**Write 3270 Device Characteristics** • Not implemented in Netrexx Pipelines.
fullscrs	**Format 3270 Device Characteristics** • Not implemented in Netrexx Pipelines.
gate	**Pass Records Until Stopped** 
gather	**Copy Records From Input Streams** • Not implemented in Netrexx Pipelines.
getfiles ***getfiles*** ***getfile*** ***getfil*** ***getfi*** ***getf*** ***get***	**Read Files** 
getovers	**Write the Contents of Objects** • Input stream 0 should contain rexx objects. The getovers stage will output the index and contents of the stem on stream 0. If output stream 1 is connected, the root is placed there. Any severed streams will cause then stage to exit. Passing a non rexx object will cause the stage to exit with return code 13. • Pipes for NetRexx only.
getstems	**Write the Contents of Members of Stems** • 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 Expresion**  • 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 expresion. Null string passes all records. • (2) Records are prefaced with records number, 10 characters, right justified. • (3) Number of records put out after a matching record. • (4) Number of records put out before and after a matching record. • (5) Inserted before a group of "before records" or the found record with "after records." • (6) Send all matching records (no numbers) to tertiary output stream, if connected. • (7) Only a count of matches is put out on the primary output stream. (Other options probably should not be used with this.)
hash ***dict***	**Read or Write a Dictionary** • Pipes for NetRexx only.
hasha ***dicta***	**Read or Write a Dictionary**
hashr ***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.
hfsquery	**Write Information Obtained from OpenExtensions into the Pipeline** • Not implemented in Netrexx Pipelines.

hfsreplace	**Replace the Contents of a File in the Hierarchical File System** • Not implemented in Netrexx Pipelines.
hfsstate	**Obtain Information about Files in the Hierarchical File System** • Not implemented in Netrexx Pipelines.
hfsxecute	**Issue OpenExtensions Requests** • Not implemented in Netrexx Pipelines.
hlasm	**Interface to High Level Assembler** • Not implemented in Netrexx Pipelines.
hlasmerr	**Extract Assembler Error Messages from the SYSADATA File** • Not implemented in Netrexx Pipelines.
hole	**Destroy Data** 
hostbyaddr *3.09*	**Resolve IP Address into Domain and Host Name**  • (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 any known and consistent manner. Could be related to a host configuration issue but googling shows odd and inconsistent results for getting around this.
hostbyname *3.09*	**Resolve a Domain Name into an IP Address**  • (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**
hostid *3.09*	**Write TCP/IP Default IP Address**  • (1) The USERid option available under CMS Pipelines is not applicable and is ignored in NetRexx Pipelines
hostname *3.09*	**Write TCP/IP Host Name**  • (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>

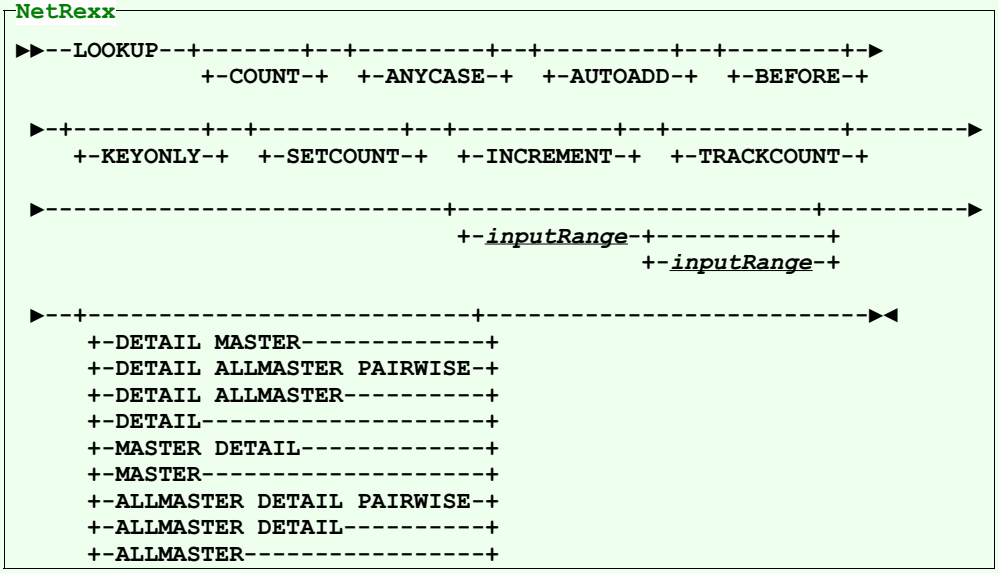
httpsplit	**Split HTTP Data Stream** • Not implemented in Netrexx Pipelines.
iebcopy	**Process IEBCOPY Data Format** • Not implemented in Netrexx Pipelines.
if	**Process Records Conditionally** • Not implemented in Netrexx Pipelines.
immcmd	**Write the Argument String from Immediate Commands** • Not implemented in Netrexx Pipelines.
insert	**Insert String in Records**  • insert a string into a record before or after the record content. Will be much more efficient than specs especially if the input is a Byte[]
inside	**Select Records between Labels** 
instore	**Load the File into a storage Buffer** • Not implemented in Netrexx Pipelines.
ip2socka	**Build sockaddr_in Structure** • Not implemented in Netrexx Pipelines.
ispf	**Access ISPF Tables** • Not implemented in Netrexx Pipelines.
jeremy	**Write Pipeline Status to the Pipeline** • Not implemented in Netrexx Pipelines.
join	**Join Records** 

joincont	**Join Continuation Lines** 
juxtapose	**Preface Record with Marker** 
ldrtbls	**Resolve a Name from the CMS Loader Tables** • Not implemented in Netrexx Pipelines.
listcat	**Obtain Data Set Names** • Not implemented in Netrexx Pipelines.
listdsi	**Obtain Information about Data Sets** • Not implemented in Netrexx Pipelines.
listispf	**Read Directory of a Partitioned Data Set into the Pipeline** • Not implemented in Netrexx Pipelines.
listpds	**Read Directory of a Partitioned Data Set into the Pipeline** • Not implemented in Netrexx Pipelines.
listzip	**List the Files in a Zipped File** 
literal	**Write the Argument String** 
locate	**Select Lines that Contain a String**  (1) Not in NetRexx Pipelines, yet.

lookup

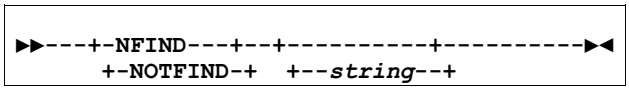
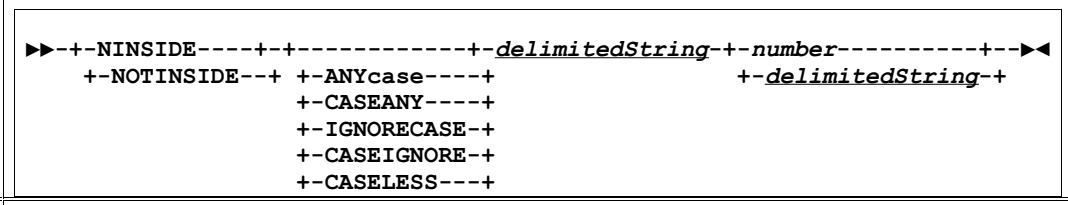
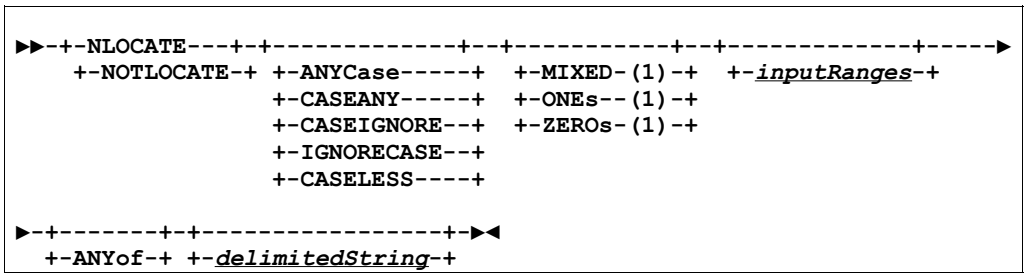
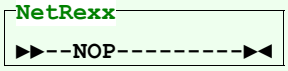
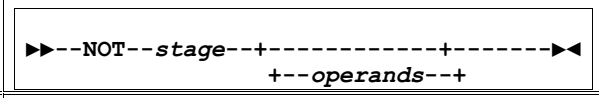
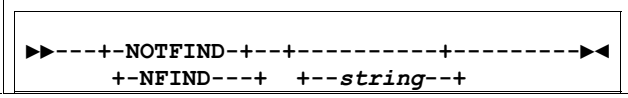
Find Records in a Reference Using a Key Field

NetRexx



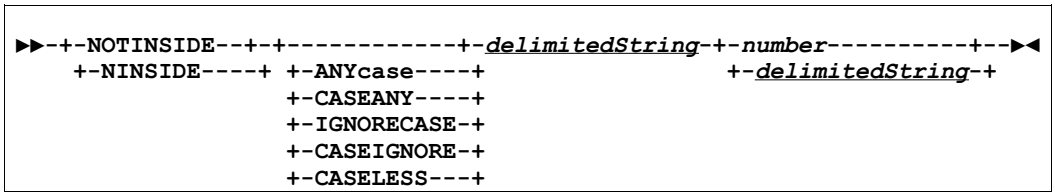
- in stream 0 are detail records
- in stream 1 are master records
- in stream 2 adds to masters
- in stream 3 delete from masters
-
- out stream 0 are matched records
- out stream 1 are unmatched detail records
- out stream 2 are unmatched or counted master records
- out stream 3 deleted masters
- out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

lookup	**Find Records in a Reference Using a Key Field** CMS ▶▶--LOOKUP--+-------+-+-----------------+-+-----------+--▶ +-COUNT-+ +-MAXcount-*number*-+ +-INCREMENT-+ +-NOPAD----+ ▶--+----------+-+------------+-+----------+-+---------+--▶ +-SETCOUNT-+ +-TRACKCOUnt-+ +-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 proprogate EOFs from output streams.
maclib	**Generate a Macro Library from Stacked Members in a COPY File** • Not implemented in Netrexx Pipelines.
mapmdisk	**Map Minidisks Into Data spaces** • Not implemented in Netrexx Pipelines.
mctoasa	**Convert CCW Operation Codes to ASA Carriage Control** • Not implemented in Netrexx Pipelines.
mdiskblk	**Read or Write Minidisk Blocks** • Not implemented in Netrexx Pipelines.
mdskrandom	**Random Access a CMS File on a Mode** • Not implemented in Netrexx Pipelines.
mdskslow	**Read, Append to, or Create a CMS File on a Mode** • Not implemented in Netrexx Pipelines.
mdskupdate	**Replace Records in a File on a Mode** • Not implemented in Netrexx Pipelines.

members	**Extract Members from a Partitioned Data Set** • Not implemented in Netrexx Pipelines.
merge	**Merge Streams** • Not implemented in Netrexx Pipelines.
mqsc	**Issue Commands to a WebSphere MQ Queue Manager** • Not implemented in Netrexx Pipelines.
nfind *notfind*	**Select Lines by XEDIT NFind Logic** 
ninside *notinside* *3.09*	**Select Records Not between Labels** 
nlocate *notlocate*	**Select Lines that Do Not Contain a String**  • (1) Not in NetRexx Pipelines, yet.
noEofBack	**Pass Records and Ignore End-of-file on Output** 
nop	**No Operation**  • Pipes for NetRexx only.
not	**Run Stage with Output Streams Inverted** 
notfind *nfind*	**Select Lines by XEDIT NFind Logic** 

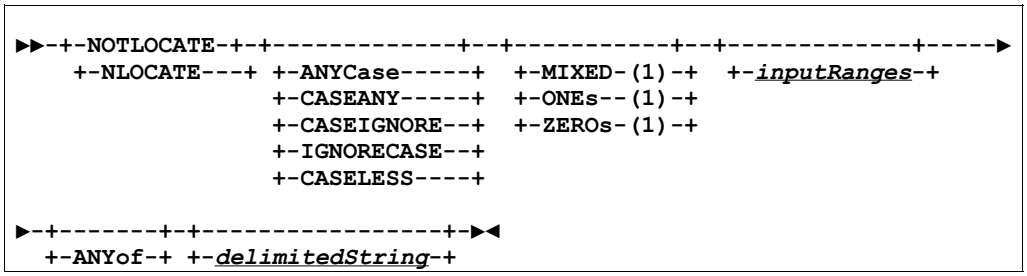
notinside
ninside

Select Records Not between Labels



notlocate
nlocate

Select Lines that Do Not Contain a String



- (1) Not in NetRexx Pipelines, yet.

nucext

Call a Nucleus Extension

- Not implemented in Netrexx Pipelines.

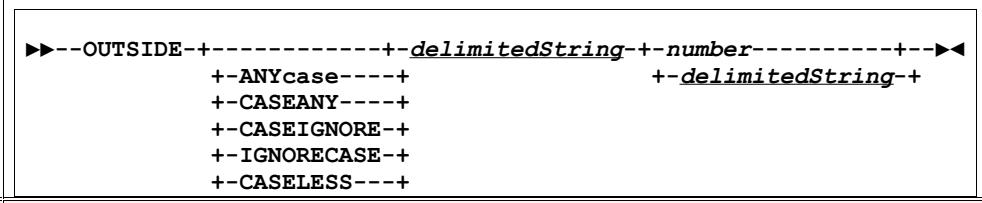
optcdj

Generate Table Reference Character (TRC)

- Not implemented in Netrexx Pipelines.

outside

Select Records Not between Labels



outstore

Unload a File from a storage Buffer

- Not implemented in Netrexx Pipelines.

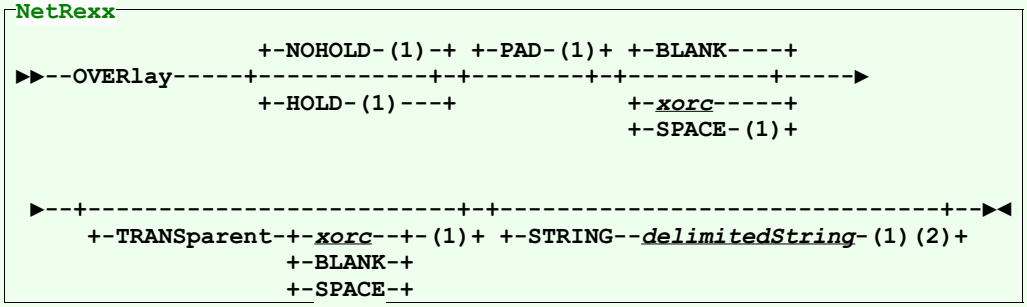
over

Write the Values of Stems

- Obsolete. Now use varover. over is now an alias for overlay..

overlay
overla
overl
over

Overlay Data from Input Streams



- 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

overlay	**Overlay Data from Input Streams**
	CMS
	`                +-BLANK-+`
	`▶▶--OVERlay-+-------+---------▶◀`
	`                +-xorc--+`

overstr	**Process Overstruck Lines** • Not implemented in Netrexx Pipelines.
pack	**Pack Records as Done by XEDIT and COPYFILE** • Not implemented in Netrexx Pipelines.

pad	**Expand Short Records**
	`        +-Right-+                                    +-BLANK-+`
	`▶▶-PAD-+-------+--+------------------+--number--+-------+--▶◀`
	`        +-Left--+  +-+--------+-MODULO-+           +-xorc--+`
	`                  +-number-+`

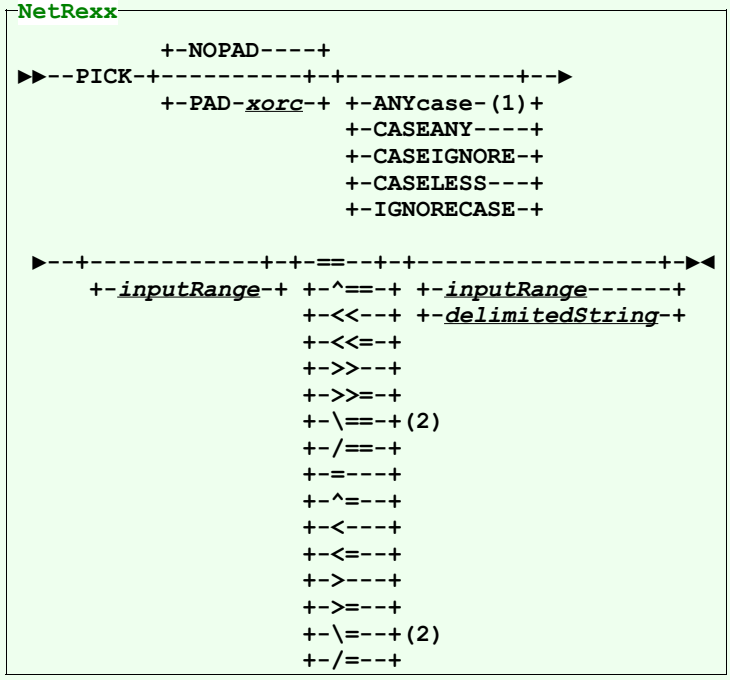
parcel	**Parcel Input Stream Into Records** • Not implemented in Netrexx Pipelines.

parse
3.09
	Rearrange Contents of Records
	NetRexx
	`▶▶--PARSE--parse_template_Dstring---output_template_Dstring--▶◀`
	• 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 $$n.
	• NetRexx Pipelines only.
	• .
	• Example:
	`parse / 2 $1 +1/ /The second letter is "$1". $$1 won't be changed./`

pause	**Signal a Pause Event** • Not implemented in Netrexx Pipelines.
pdsdirect	**Write Directory Information from a CMS Simulated Partitioned Data Set** • Not implemented in Netrexx Pipelines.

pick	**Select Lines that Satisfy a Relation**

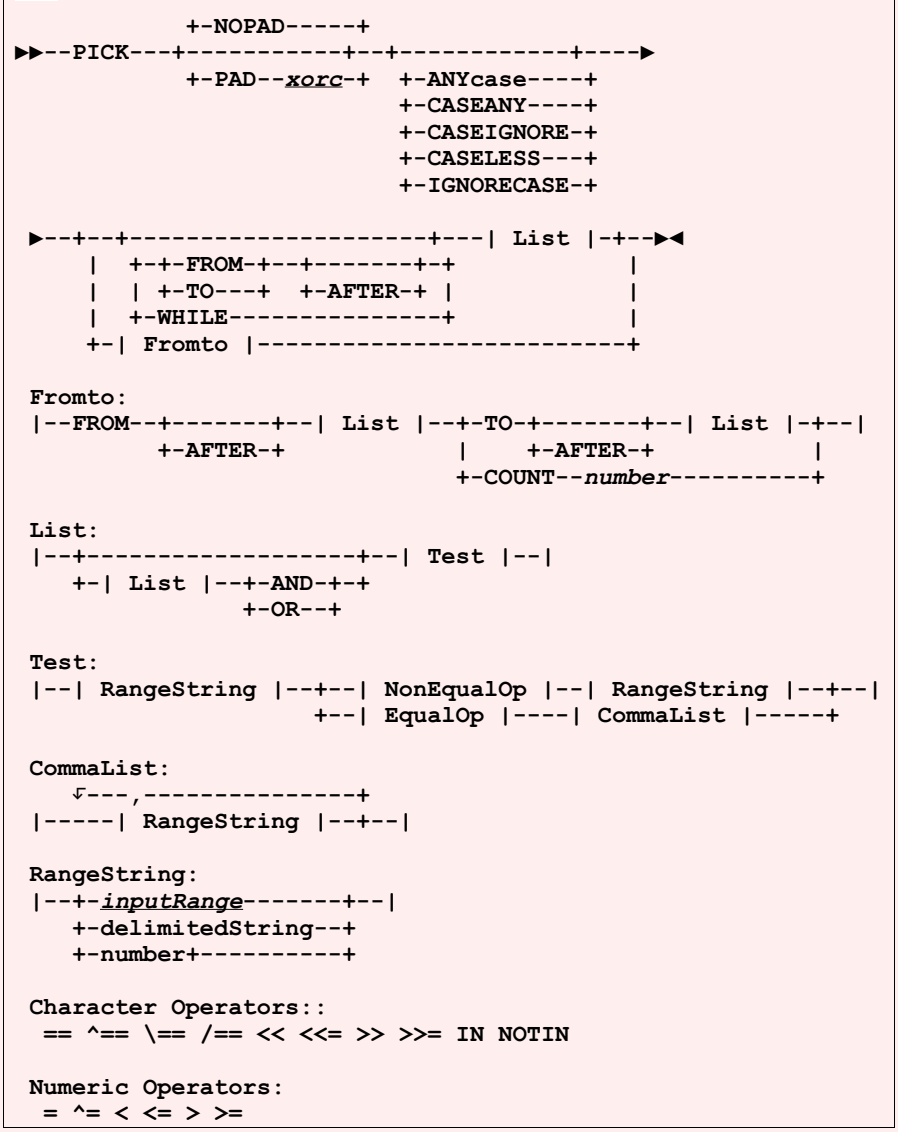
NetRexx



- (1) Can be before PAD/NOPAD. Depreciated.
- (2) The backslash (\) may need to be escaped, doubled, in some systems shells.

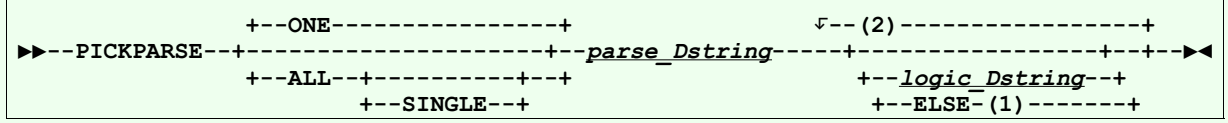
pick	**Select Lines that Satisfy a Relation**

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 put out 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 ouput 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 PARSE template.

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

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

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

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

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

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

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

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

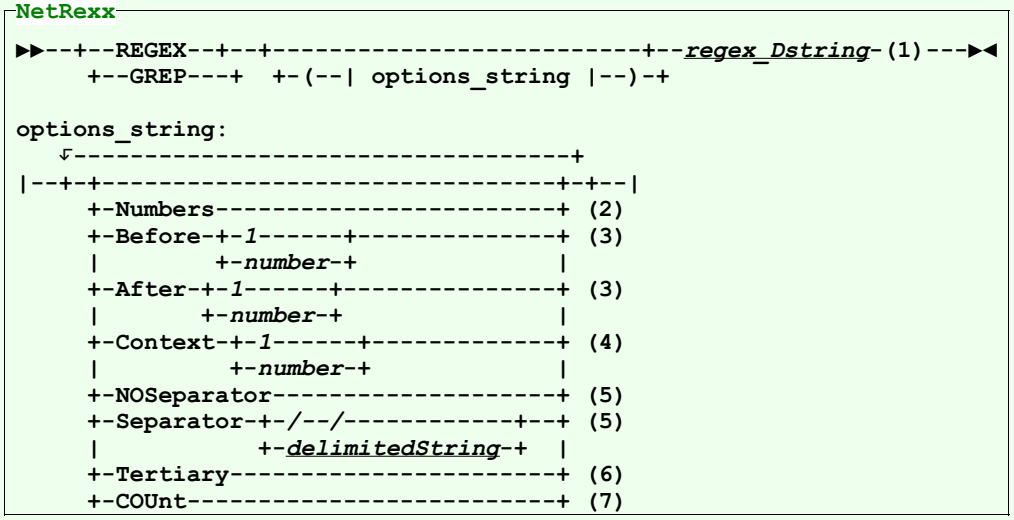
pipcmd	**Issue Pipeline Commands** • Not implemented in Netrexx Pipelines.
pipestop	**Terminate Stages Waiting for an External Event** • Not implemented in Netrexx Pipelines.
polish	**Reverse Polish Expression Parser** • Not implemented in Netrexx Pipelines.
predselect	**Control Destructive Test of Records** • Not implemented in Netrexx Pipelines.
preface	**Put Output from a Device Driver before Data on the Primary Input Stream** • Not implemented in Netrexx Pipelines.

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

regex
grep
3.09

Select Lines by a Regular Expresion

NetRexx



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

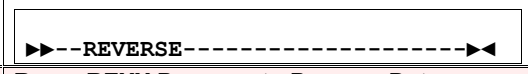
retab

Replace Runs of Blanks with Tabulate Characters

- Not implemented in Netrexx Pipelines.

reverse

Reverse Contents of Records



rexx

Run a REXX Program to Process Data

- Not implemented in Netrexx Pipelines.

rexxvars

Retrieve Variables from a REXX or CLIST Variable Pool

- Not implemented in Netrexx Pipelines.

runpipe

Issue Pipelines, Intercepting Messages

- Not implemented in Netrexx Pipelines.

scm

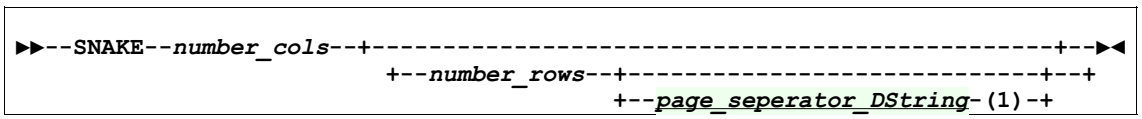
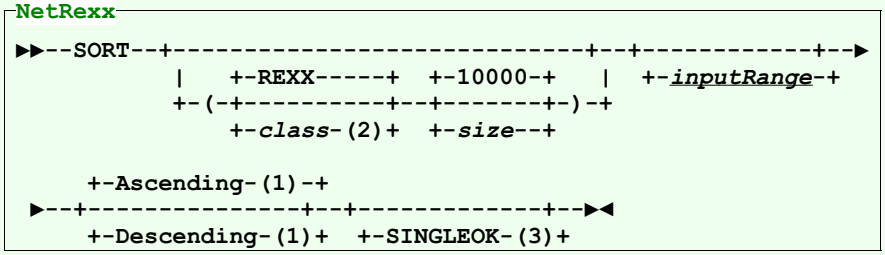
Align REXX Comments

- Not implemented in Netrexx Pipelines.

sec2greg

Convert Seconds Since Epoch to Gregorian Timestamp

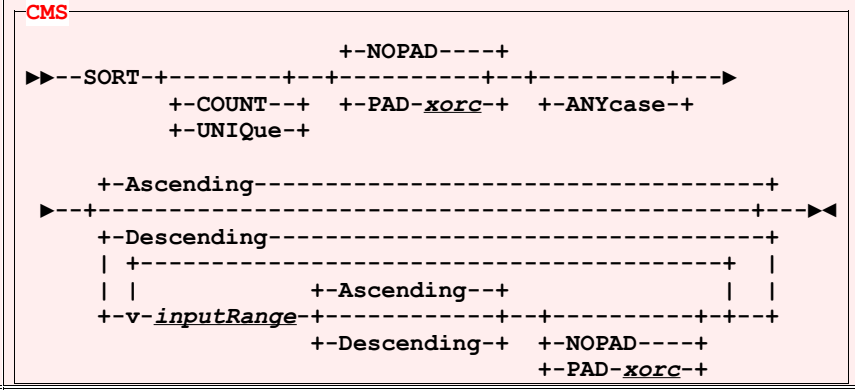
- Not implemented in Netrexx Pipelines.

serialize	**Convert Objects to a Single Text String** NetRexx ***limited information available*** • {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
sfsback	**Read an SFS File Backwards** • Not implemented in Netrexx Pipelines.
sfsdirectory	**List Files in an SFS Directory** • Not implemented in Netrexx Pipelines.
sfsrandom	**Random Access an SFS File** • Not implemented in Netrexx Pipelines.
sfsupdate	**Replace Records in an SFS File** • Not implemented in Netrexx Pipelines.
snake *3.09*	**Build Multicolumn Page Layout**  • (1) NetRexx Pipelines only. Appears first, last, and between pages. Avoid \ as escape terms maybe added in the future. \n for newline is OK. Your system may require \\n .
socka2ip	**Format sockaddr_in Structure** • Not implemented in Netrexx Pipelines.
sort	**Order Records** NetRexx  • (1) May come before inputRange, for backwards compatability. • (2) Requires that you implement another sortClass with a name begining with 'sort' • (3) Suppresses error message if only one record to sort for Rexx objects. • Uses *sortClass* class as Interface Class for Generic Sort Objects and *sortRexx* class to Sort Rexx Text Objects

sort

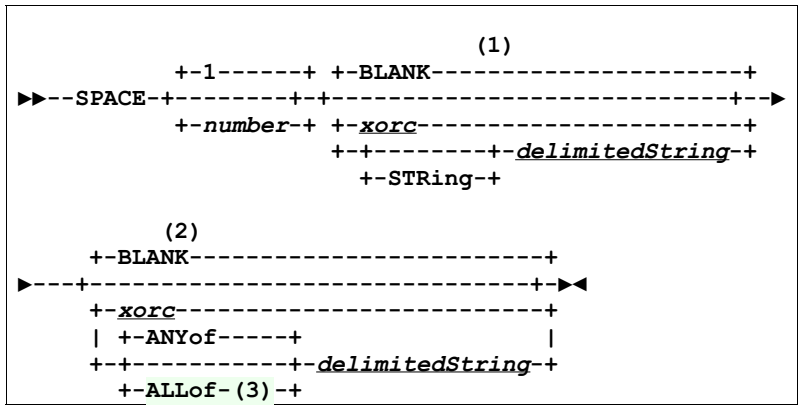
Order Records

CMS



space
3.09

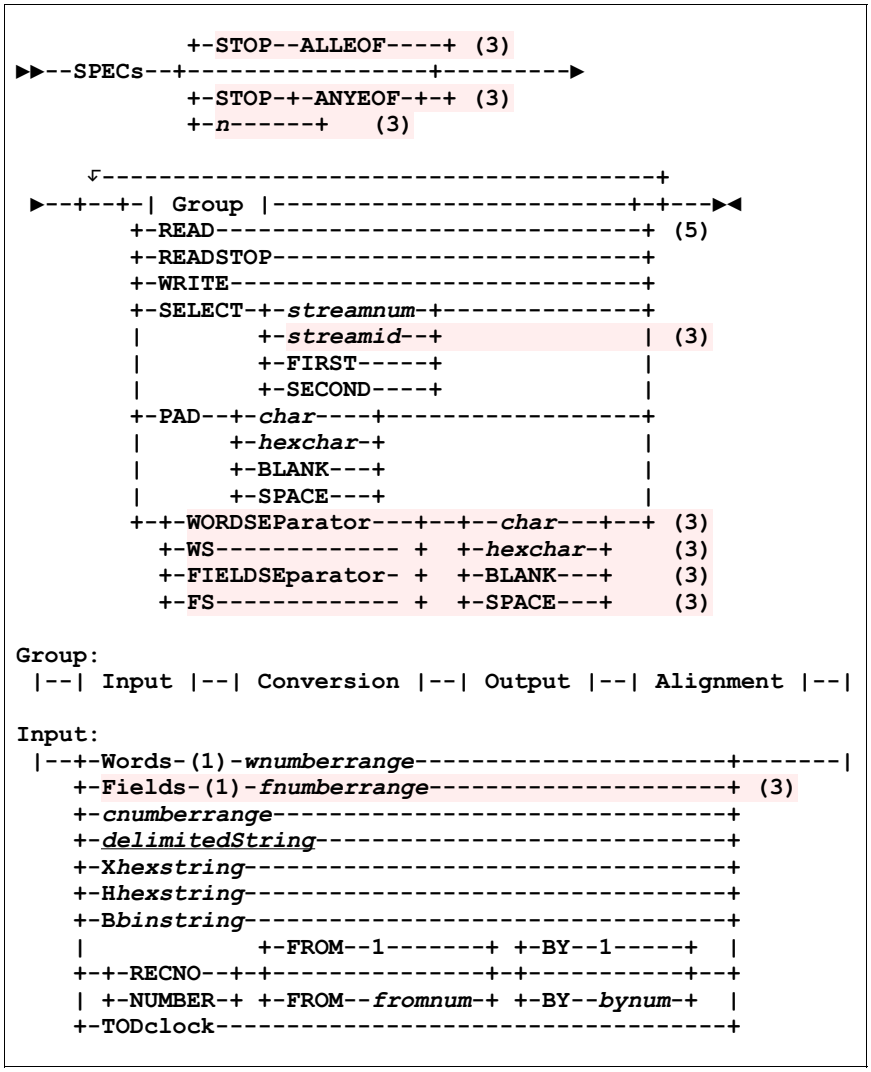
Space Words Like REXX



- (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

spec
specs

Rearrange Contents of Records


+-STOP--ALLEOF----+ (3)
▶▶--SPECs--+----------------+--------▶
+-STOP-+-ANYEOF-+-+ (3)
+-n------+ (3)
▶--+--+-| Group |-------------------------+-+---▶◀
+-READ-------------------------------+ (5)
+-READSTOP---------------------------+
+-WRITE--------------------------------+
+-SELECT-+-streamnum-+--------------+
| +-streamid--+ | (3)
| +-FIRST-----+ |
| +-SECOND----+ |
+-PAD--+-char----+------------------+
| +-hexchar-+ |
| +-BLANK---+ |
| +-SPACE---+ |
+-+-WORDSEParator---+--+--char---+--+ (3)
+-WS------------ + +-hexchar-+ (3)
+-FIELDSEparator- + +-BLANK---+ (3)
+-FS------------ + +-SPACE---+ (3)
Group:
|--| Input |--| Conversion |--| Output |--| Alignment |--|
Input:
|--+-Words-(1)-wnumberrange----------------------+-------|
+-Fields-(1)-fnumberrange---------------------+ (3)
+-cnumberrange-------------------------------+
+-delimitedString----------------------------+
+-Xhexstring---------------------------------+
+-Hhexstring---------------------------------+
+-Bbinstring---------------------------------+
| +-FROM--1-------+ +-BY--1-----+ |
+-+-RECNO--+-+---------------+-+-----------+--+
| +-NUMBER-+ +-FROM--fromnum-+ +-BY--bynum-+ |
+-TODclock----------------------------------+

```
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--+
```

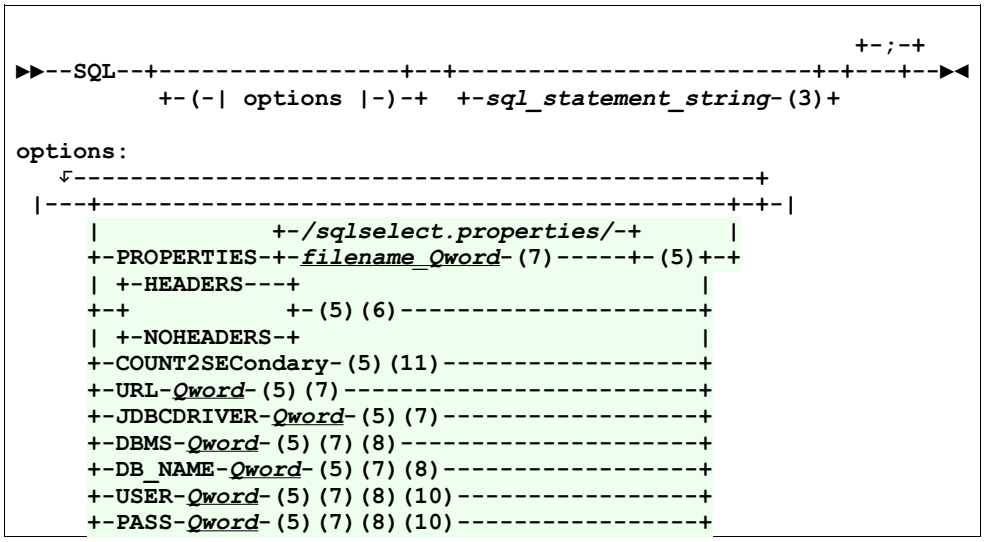
spill	**Spill Long Lines at Word Boundaries** • Not implemented in ...
split	**Split Records Relative ...**

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

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

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

```
Ranges (cnumberrange, fnumberrange (3), wnumberrange):
 |--+-snumber-+--(2)--+-------------------------+--|
    +---*-----+       +--.-----(2)----number----+
                      +- - -+--(2)--+-snumber-+-+
                      +--;--+       +---*-----+
```

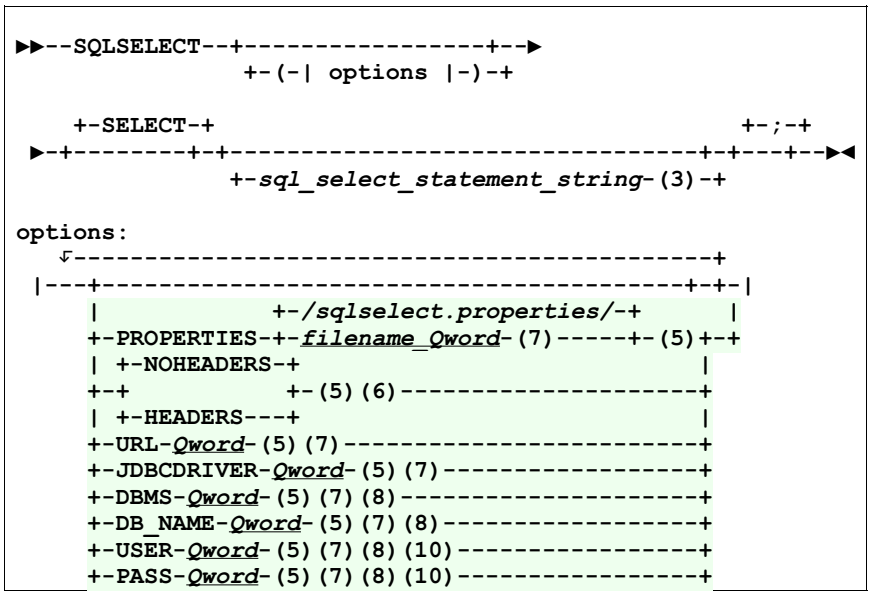
- (1) Blanks are optional in this position.
- (2) Blanks are not allowed here.
- (3) CMS only. Not yet implemented in NetRexx Pipelines
- (4) NetRexx Pipelines only. Not yet implemented in CMS
- (5) READ is giving the same output as READSTOP when the streams are different length.
- [6] This senses if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

sql *3.09*	**Interface to SQL**



- uses jdbc to select from any jdbc enabled dbms
- properties file (sqlselect.properties default) is read from the secondary input stream to find jdbcdriver name, url, user, pass
- sample properties file:
-
```
#JDBC driver name
#Tue Feb 03 23:29:43 GMT+01:00 1998
jdbcdriver=com.imaginary.sql.msql.MsqlDriver
url=jdbc:msql://localhost:1114/TESTDB
# the following are not needed for some DBMS, ex: SQLite
user=db_user_name
pass=password_for_db
```
-
- if this file is not found default (compiled in) values are used
- (1) when using a sql select * (all columns) from the 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 implyed 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. These are used in place of URL and JDBCDRIVER. SQLite is the only one tested as of 8/15/20.
- (9) the SQLSELECT stage uses HEADERS as the default.
- (10) USER & PASS are needed for some DBMSs and not others, ex. SQLite.
- (11) the count or other output from non-select statements goes to the secondary output stream if connected, or is discarded. Otherwise it goes to the primary.
-
- Priority order for URL, JDBCDRIVER and DBMS, DB_NAME (first one found rules):
  1. option in the SQL command string
  2. from secondary input stream
  3. from "sql.properties" file or from file specified by PROPERTIES option
  4. Builtin

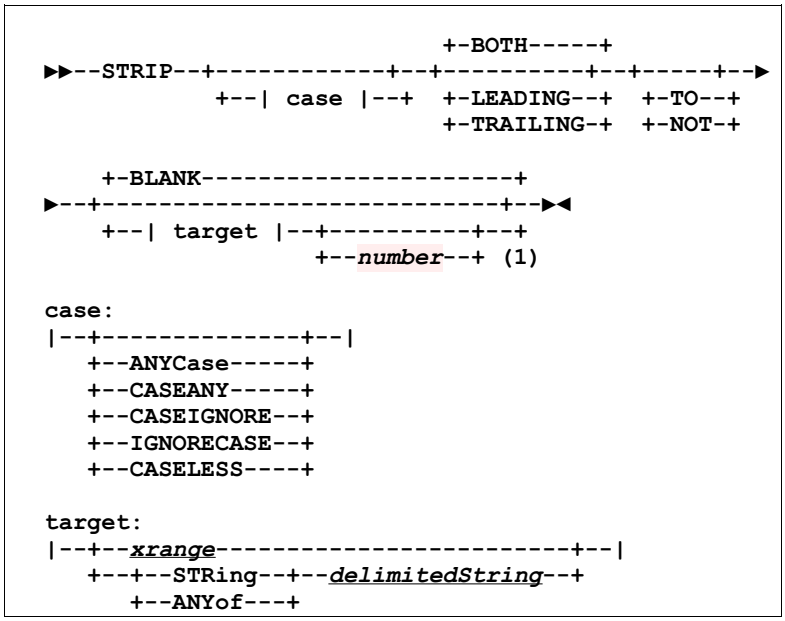
sqlcodes	**Write the last 11 SQL Codes Received** • Not implemented in Netrexx Pipelines.

sqlselect	**Query a Database and Format Result**  • (1) when using a sqlselect * (all columns) from the commandline, quote the query as in java pipes.compiler (query) "sqlselect * 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. These are 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.
state	**Provide Information about CMS Files** • Not implemented in Netrexx Pipelines.
state	**Verify that Data Set Exists** • Not implemented in Netrexx Pipelines.
statew	**Provide Information about Writable CMS Files** • Not implemented in Netrexx Pipelines.

stem	**Retrieve or Set Variables in a REXX or CLIST Variable Pool** NetRexx ▶▶--STEM--*stem*--------------▶◀
stem	**Retrieve or Set Variables in a REXX or CLIST Variable Pool** CMS ▶▶--STEM--*stem*--+----------+-+--------+-+----------+-▶ +-PRODUCER-+ +-*number*-+ +-NOMSG233-+ +-MAIN-----+ +-SYMBOLIC-+ ▶-+----------+-+--------------+-----▶◀ +-DIRECT---+ +-APPEND-------+ +-FROM--*number*-+
stfle	**Store Facilities List** • Not implemented in Netrexx Pipelines.
storage	**Read or Write Virtual Machine Storage** • Not implemented in Netrexx Pipelines.
strasmfind	**Select Statements from an Assembler File as XEDIT Find** • Not implemented in Netrexx Pipelines.
strasmnfind	**Select Statements from an Assembler File as XEDIT NFind** • Not implemented in Netrexx Pipelines.
strfind	**Select Lines by XEDIT Find Logic** ▶▶----STRFIND--+------------+--*delimitedString*----------▶◀ +-ANYcase----+ +-CASEANY----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS---+ • Not implemented in Netrexx Pipelines.
strfrlabel *strfrlabe* *strfrlab* *strfrlab*	**Select Records from the First One with Leading String** +--STRFROMLABEL--+ +-INCLUSIVe-+ ▶▶--+--STRFRLABel----+-+------------+-+----------+-*delimitedString*--▶◀ +-ANYcase----+ +-EXCLUSIVe-+ +-CASEANY----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS---+
strfromlabel *strfrlabel* *strfrlabe* *strfrlab*	**Select Records from the First One with Leading String** +--STRFROMLABEL--+ +-INCLUSIVe-+ ▶▶--+--STRFRLABel----+-+------------+-+----------+-*delimitedString*--▶◀ +-ANYcase----+ +-EXCLUSIVe-+ +-CASEANY----+ +-IGNORECASE-+ +-CASEIGNORE-+ +-CASELESS---+

strip

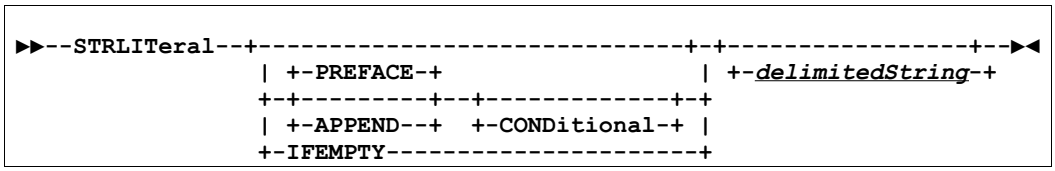
Remove Leading or Trailing Characters



- (1) Not implemented in Netrexx Pipelines.

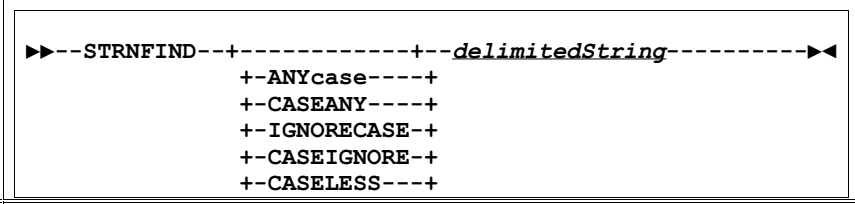
strliteral

Write the Argument String



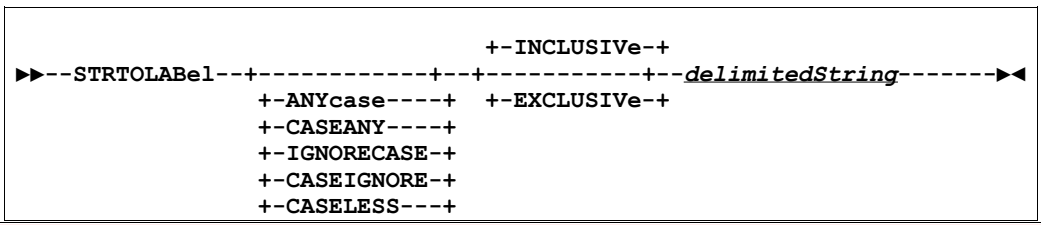
strnfind

Select Lines by XEDIT NFind Logic



strtolabel
strtolabe
strtolab

Select Records to the First One with Leading String



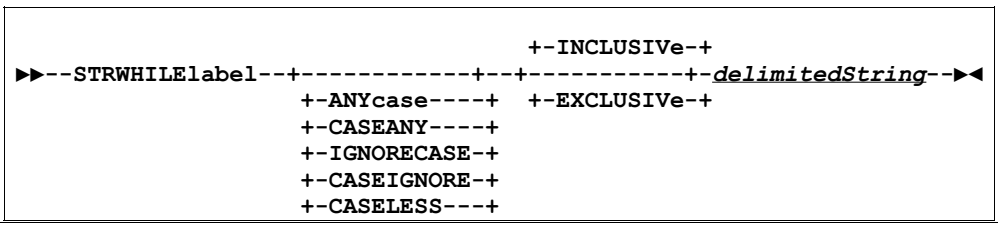
structure

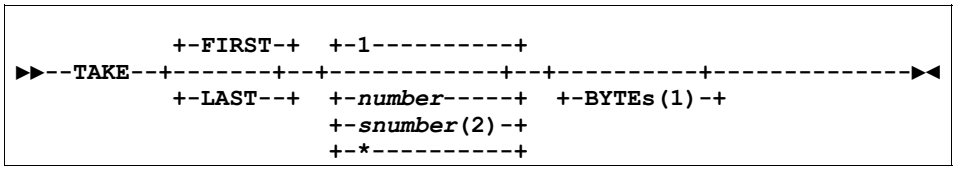
Manage Structure Defnitions

- Not implemented in Netrexx Pipelines.

strwhilelable
strwhilelabl
strwhilelab
strwhilela
strwhilel
strwhile
3.09

Select Run of Records with Leading String



stsi	**Store System Information** • Not implemented in Netrexx Pipelines.
subcom	**Issue Commands to a Subcommand Environment** • Not implemented in Netrexx Pipelines.
substring	**Write substring of record** • Not implemented in Netrexx Pipelines.
synchronise ***synchronize***	**Synchronise Records on Multiple Streams** • Not implemented in Netrexx Pipelines.
synchronize ***synchronise***	**Synchronise Records on Multiple Streams** • Not implemented in Netrexx Pipelines.
sysdsn	**Test whether Data Set Exists** • Not implemented in Netrexx Pipelines.
sysout	**Write System Output Data Set** • Not implemented in Netrexx Pipelines.
sysvar	**Write System Variables to the Pipeline** • Not implemented in Netrexx Pipelines.
take	**Select Records from the Beginning or End of the File**  • (1) CMS must be BYTES • (2) Not CMS; NetRexx Pipelines: minus reverses first/last
tape	**Read or Write Tapes** • Not implemented in Netrexx Pipelines.
tcpchsum	**Compute One's complement Checksum of a Message** • Not implemented in Netrexx Pipelines.
tcpclient	**Connect to a TCP/IP Server and Exchange Data** • Simple tcpclient implementation. The options implemented are similar to the CMS definition. ◦ linger - wait a bit before terminating the last read (units SECONDS) ◦ timeout - wait this long before timing reads out (units MS) ◦ deblock - If deblock is omitted a copy stage is used. ◦ group - similar to CMS. A delimited string containing a stage is expected. You can use a run of stages, but its is dangerous since you to know the stage sep character being used... ◦ greeting - expect a greeting message and discard it ◦ nodelay - use the nodelay option ◦ keepalive - enable keep alive socket option ◦ oneresponse - synchronize cmds/replys

tcpdata

Read from and Write to a TCP/IP Socket

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

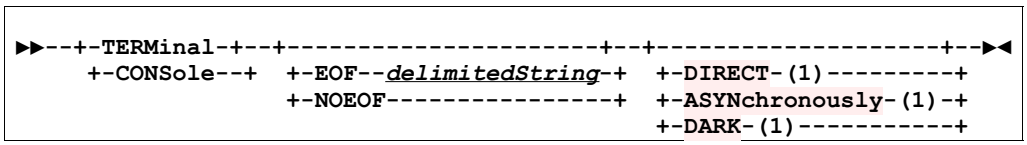
tcplisten

Listen on a TCP Port

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

terminal
termina
termina
termin
termi
term
console
consol
conso
cons
cons

Read or Write the Terminal in Line Mode



- (1) CMS Pipelines Only.

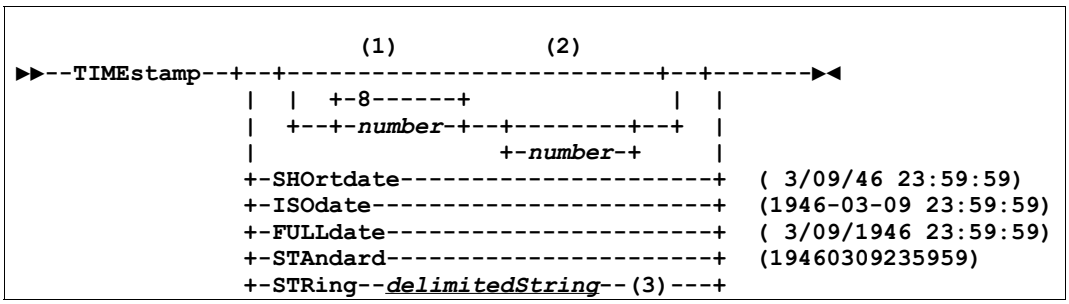
threeway

Split record three ways

- Not implemented in Netrexx Pipelines.

timestamp

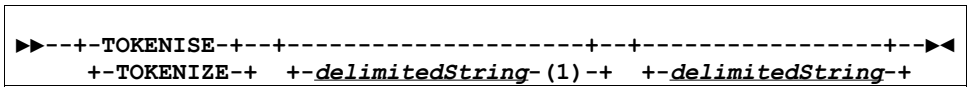
Prefix the Date and Time to Records



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

tokenise
tokenize

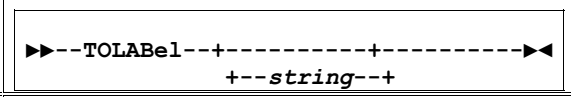
Tokenise Records



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

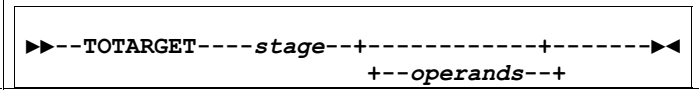
tolabel
tolabe
tolab

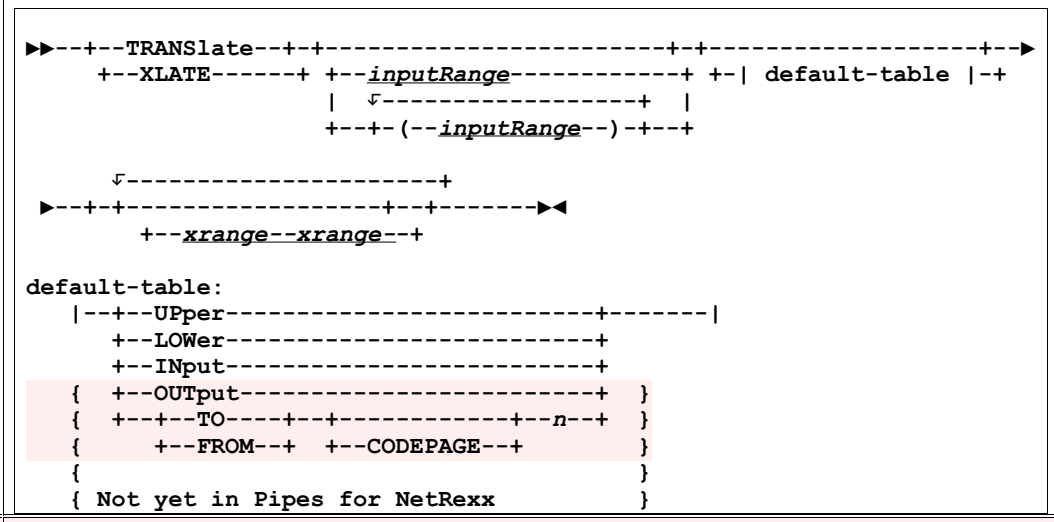
Select Records to the First One with Leading String



totarget

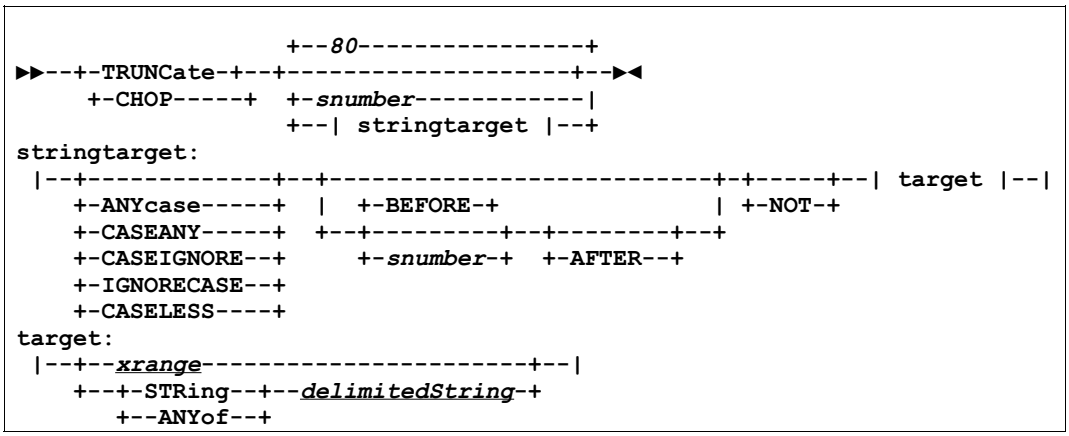
Select Records to the First One Selected by Argument Stage



trackblock	**Build Track Record** • Not implemented in Netrexx Pipelines.
trackdeblock	**Deblock Track** • Not implemented in Netrexx Pipelines.
trackread	**Read Full Tracks from ECKD Device** • Not implemented in Netrexx Pipelines.
tracksquish	**Squish Tracks** • Not implemented in Netrexx Pipelines.
trackverify	**Verify Track Format** • Not implemented in Netrexx Pipelines.
trackwrite	**Write Full Tracks to ECKD Device** • Not implemented in Netrexx Pipelines.
trackxpand	**Unsquish Tracks** • Not implemented in Netrexx Pipelines.
translate ***translat*** ***translate*** ***transl*** ***trans*** ***xlate***	**Transliterate Contents of Records** 
trfread	**Read a Trace File** • Not implemented in Netrexx Pipelines.

truncate
truncat
trunca
trunc
chop

Truncate the Record



tso

Issue TSO Commands, Write Response to Pipeline

- Not implemented in Netrexx Pipelines.

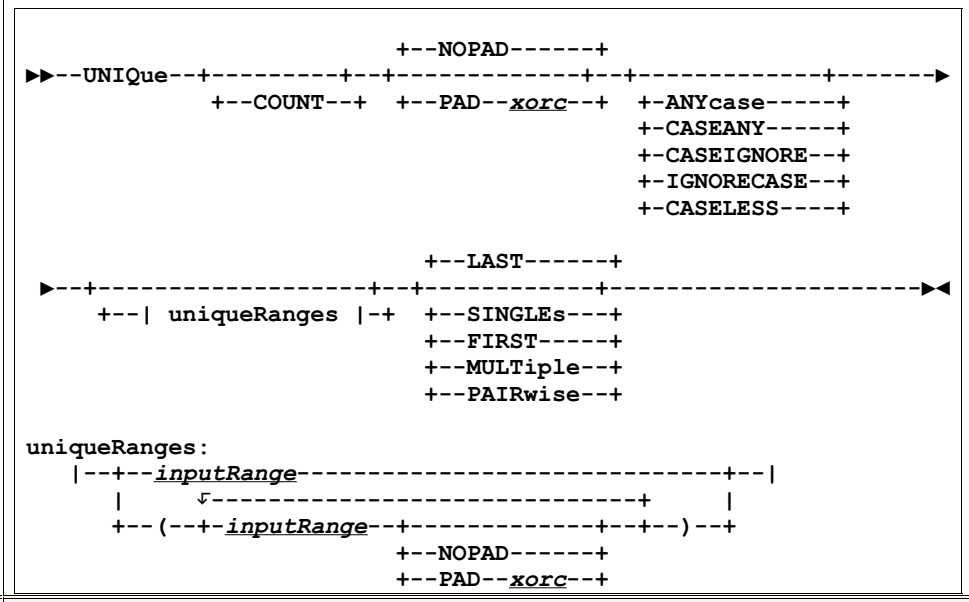
udp

Read and Write an UDP Port

- Not implemented in Netrexx Pipelines.

unique
uniqu
uniq

Discard or Retain Duplicate Lines



unpack

Unpack a Packed File

- Not implemented in Netrexx Pipelines.

untab

Replace Tabulate Characters with Blanks

- Not implemented in Netrexx Pipelines.

update

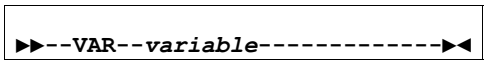
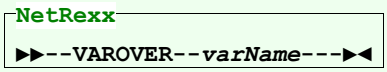
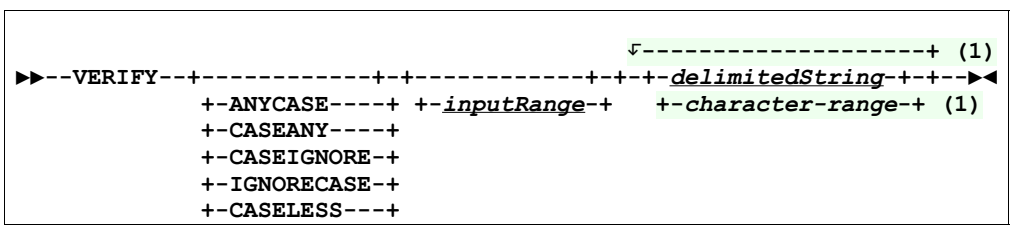
Apply an Update File

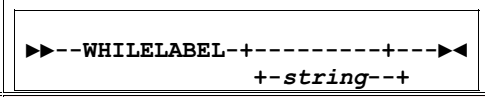
- Not implemented in Netrexx Pipelines.

urldeblock

Process Universal Resource Locator

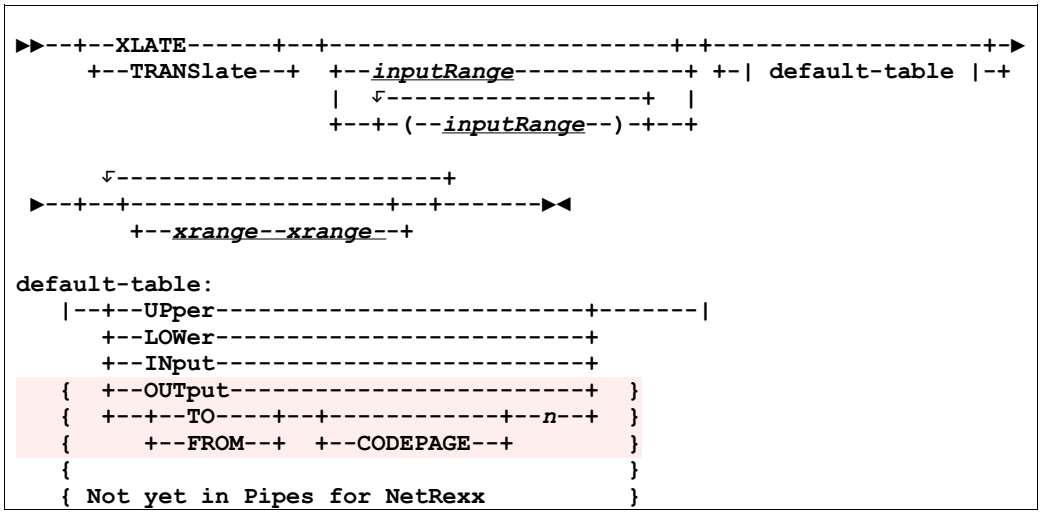
- Not implemented in Netrexx Pipelines.

uro	**Write Unit Record Output** • Not implemented in Netrexx Pipelines.
utf	**Convert between UTF-8, UTF-16, and UTF-32** • Not implemented in Netrexx Pipelines.
var	**Retrieve or Set a Variable in a REXX or CLIST Variable Pool**  • Pipes for NetRexx: this can only read vars
vardrop	**Drop Variables in a REXX Variable Pool** • Not implemented in Netrexx Pipelines.
varfetch	**Fetch Variables in a REXX or CLIST Variable Pool** • Not implemented in Netrexx Pipelines.
varload	**Set Variables in a REXX or CLIST Variable Pool** • Not implemented in Netrexx Pipelines.
varover *3.09*	**Write the Values of Stems**  • NetRexx Pipelines only; not CMS Pipelines
varset	**Set Variables in a REXX or CLIST Variable Pool** • Not implemented in Netrexx Pipelines.
vchar	**Recode Characters to Different Length** • Not implemented in Netrexx Pipelines.
vector	**Read or Write an Array of Vectors** • Pipes for NetRexx only.
vectora	**Add to an Array of Vectors** • Pipes for NetRexx only.
vectorr	**Read From an Array of Vectors** • Pipes for NetRexx only.
vectorw	**Write to an Array of Vectors** • Pipes for NetRexx only.
verify *3.09*	**Verify that Record Contains only Specified Characters**  • (1) NetRexx Pipelines only • (1) Examples: A-Z 0-9 c-g a4-ba; 16-bit Unicode characters or hex numbers • (1) Any number greater than zero, any order, of delimitdStrings and character-ranges are allowed.

vmc	**Write VMCF Reply** • Not implemented in Netrexx Pipelines.
vmcdata	**Receive, Reply, or Reject a Send or Send/receive Request** • Not implemented in Netrexx Pipelines.
vmclient	**Send VMCF Requests** • Not implemented in Netrexx Pipelines.
vmclisten	**Listen for VMCF Requests** • Not implemented in Netrexx Pipelines.
waitdev	**Wait for an Interrupt from a Device** • Not implemented in Netrexx Pipelines.
warp	**Pipeline Wormhole** • Not implemented in Netrexx Pipelines.
warplist	**List Wormholes** • Not implemented in Netrexx Pipelines.
whilelabel *3.09*	**Select Run of Records with Leading String** 
wildcard	**Select Records Matching a Pattern** • Not implemented in Netrexx Pipelines.
writepds	**Store Members into a Partitioned Data Set** • Not implemented in Netrexx Pipelines.
xab	**Read or Write External Attribute Buffers** • Not implemented in Netrexx Pipelines.
xedit	**Read or Write a File in the XEDIT Ring** • Not implemented in Netrexx Pipelines.

xlate
translate
translat
transla
transl
trans

Transliterate Contents of Records



xmsg

Issue XEDIT Messages

- Not implemented in Netrexx Pipelines.

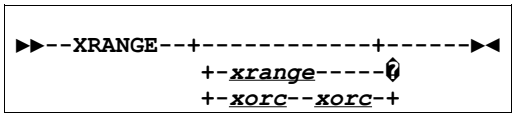
xpndhi

Expand Highlighting to Space between Words

- Not implemented in Netrexx Pipelines.

xrange
3.09

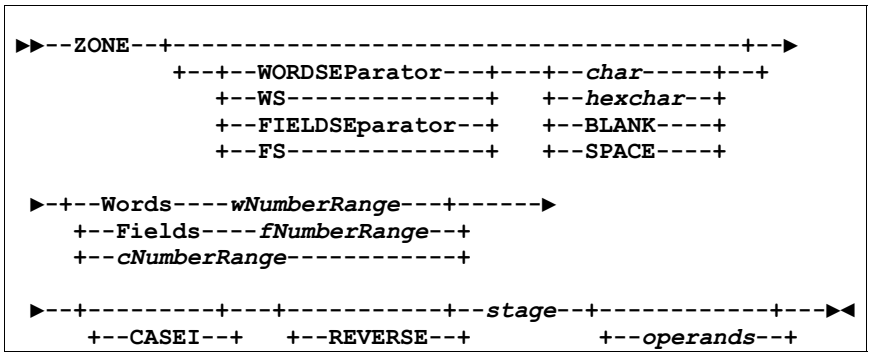
Write a Range of Characters



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

zone

Run Selection Stage on Subset of Input Record



20

Appendix A

```
.50  - Released May 30, 1999
         - Fixed a stall occurring when interrupted threads, with the interrupt
           caught by ThreadPool, were reused.
         - Fixed a thread safety problem in ELASTIC
         - Improved the timeout options in TCPDATA and TCPCLIENT, they also
           byte[] instead of strings.  This was done since converting to and
           from strings sometimes scrambles binary data (more research on
           encodings...)
         - Changed DELBLOCK it now handles byte[] to help keep tcpdata and
           tcpclient efficient.  The EOF option was broken, its fixed now.
         - Changed DISKR, DISKW and DISKA to handle byte[] when using streams.
         - Added INSERT which handles byte[].  This should be used instead of
           SPECS to add LF or CR .
         - Changes SERIALIZE to use byte[].
   0.49  - Released May 21, 1999
         - compiled with 1.2.1 and NetRexx 1.148
         - Added preliminary support added to .njp compiler for files containing
           java source!  See the (some what messy) java samples in vectort1.njp,
           overtest.njp and addtest4.njp
         - Added code to generate a dummy .nrx file containing the public class
           in a .java file.  This allows NetRexx to compile class that depend on
           the java source.
         - Modified sort to accept arguements in the same order as CMS
         - Fixed rc logic in drop stage
         - Fixed shortcut code for {n} where n is numeric.
   0.48  - Released May 16, 1999
         - Fixed a (nasty) bug involving reusing pipe objects.
         - Added the reuse() method to the stage class.  To use it override
           it in your stage.  It was added so there was a foolproof way to
           reset a stage when its pipe object is reused.  (doSetup is intended
           for use with dynamic arguements in call or added pipes)
         - Added the cont option and defaulted it to comma.
         - fixed return code logic in some stages and in selectInput/Output
         - Added the Emsg methods
         - Added arguement debug option (128)
         - There are no more final methods
         - Much improved error reporting from stages via new Emsg method
   0.47  - Released Jan 3, 1999
```

```
      - recompiled with 1.1.7A and netrexx 1.148
      - UNIQUE repaired?
      - Added stages to acess java objects easily
        VECTOR, VECTORR, VECTORW, VECTORA for java.util.vector
        ARRAY, ARRAYR, ARRAYW, ARRAYA for Object[]
        HASH, HASHR, HASHW, HASHA for java.util.Hashtable
        DICT, DICTR, DICTW, DICTA for java.util.Dictionary
        The hash stages mostly map directly to DICT stages.  The exception
        is HASHW which uses the clear() method of Hashtable.
      - Modified LITERAL to be able to put any object into a pipe
      - Modified pipe package to store arguements in a hashtable instead of
        a rexx stem - arguements can now be of any class.  Use the arg(null)
        method to get an object arguement.
0.46a - Released Oct 14, 1998
      - recompiled with 1.1.7
      - TCPLISTEN now supports an input stream to be used to pace accepts
0.46  - Released Sept 20, 1998
      - COMMAND, CHANGE, FILE, LOCATE, DROP, LOOKUP, TCPCLIENT, TCPLISTEN
        SQLSELECT, CONSOLE, TCPDATA, NOEOFBACK improved.
      - Jeff improved the testing process with the addition of the COMPARE
        stage, he also upgraded many of the tests.
      - Added the buildtests pipe, it builds a test script to be run with:
        test > output < console.data
      - Unexpected exceptions should no longer hang pipes
0.45  - Released Sept 9, 1998
      * Recompile all your stages.  To fix a commit problem I had to
        change the _stage interface class...
      - tcpclient restart problems with oneresp active fixed.
      - commit now returns the current return code of the pipe.
      - fixed minor errors in tcpclient and diska.
0.44  - Released Sept 8, 1998
      * a recompile of pipes using STEM is required
      - smart DISK, FILE and STEM stages now exist.
      - Made to and from synonoms for in and out in REXX and STRING stages.
      - Added stream option to DISKR and DISKW to read raw streams.
      - Added DISKSLOW and SERIALIZE stages.
      - Now DISK, DISKR, DISKW, DISKA and DISKSLOW have FILE synonyms.
      - Deadlock detection improvements.
      - TCPDATA & TCPCLIENT optimized once again.
      - selectAnyInput could deadlock - fixed.
      - interrupting a pipe now kills it - use this with care (ie. kill -9)
      - Pseudo methods njpRC() and njpObject() are reconized by the pipes
        compiler and return the pipe's RC or object respectivily.
0.43  - Released August 30, 1998
      - Fixed deadlock dection to see commit deadlocks.
      - Added rest of code to handle improved StageError logic.
      - Added stage templates (template*.nrx) in the njpipes directory.
```

```
      - Added a debug flag (64) to trace all StageError rasied by the
        stage class.
0.42  - Not released
      * A recompile of pipes using TCPCLIENT, TCPDATA is required.
      * A recompile of pipes using REXX, STRING, ZONE, CASEI is recommended.
      - Updated the comments in _stage to reflect the possible StageError
        and return codes that can be issued.
      - Added the DEBLOCK stage and reworked TCPDATA, TCPCLIENT & GATE.
      - Improved eofReport processing and added a new option 'either' that
        will trigger a StageError when any stream, input or output, severs.
      - Fixed variable subsitution so multiple variables passed to a stage
        will work.
      - Added the ability to pass thru arguements to callpipe and addpipe.
      - Fixed a problem with some StageExits requiring stage_reset methods.
      - Added a function to utils to help assign smarter name to classes
        generated by StageExits.
      - Added counter method to stage.  use to count external waits so
        deadlock/stall detection is not fooled.
0.41  - Released August 23, 1998
      * removed OBJ2REXX, OBJ2STRING stages, use REXX and STRING stage
        modifiers.
      * pipes using TCPDATA, TCPCLIENT & LOOKUP should be recompiled
      - exhanced REXX stage modifier via an object2rexx improvement in
        pipes/utils.nrx
      - optimized ThreadPool startup times.  No setName and only use
        setPriority when its required.
      - made it possible to optimized stage startup time when arguements
        are static.  See TCPDATA, TCPCLIENT & LOOKUP
      - faq.txt enhanced
0.40  - Released August 14, 1998
      * All pipes MUST be recompiled.  Old pipe class files will stall.
      - OBJ2REXX is depreciated and will be removed, use the REXX stage.
      - added REXX and STRING stages to convert objects entering and leaving
        a stage to rexx or string.  Inorder to avoid nasty class conflicts,
        REXX and STRING are implemented in _rexx and _string.  The compler
        adds the '_' when necessary (any stage can use this feature).
      - fixed an intermitant stall in callpipe (was completing too fast :-)
      - fixed a stall occuring between shortStreams and COMMAND
      - optimized pipe startup time in pipe.class and via the compiler.
      - optimized rc, commit, deadlock, threadpool code
0.39  - Released August 9, 1998
      - WAIT_COMMIT and WAIT_ANY are now used in the call/addpipe logic
      - callpipe was not notifiing its pipe when ending leading to an
        very intermitant hang.
0.38  - Released August 3, 1998
      * All your stages must be recompiled.  Recompile your pipes to
        exploit the pipe & thread pool performance improvements.
```

```
      - fixed and optimized commit logic.
      - implement a pool for pipes to decrease overhead.
      - implement a pool for threads to decrease overhead.
      - compiler fix to proprogate return codes from stageExits (thanks Jeff).
      - signal StageError('...  in all stageExits modified to
        signal StageError(13,'Error - 'pInfo' - ...
      - UNIQUE stage added by Jeff.  It exploits stageExit.
      - COMMAND stage was not starting its threads correctly.
      - SORTs in different pipes could corrupt each other.  Thanks René,
0.37  - Released July 25, 1998
      * A recompile of pipes using SORT is required
      - added NOEOFBACK, TOTARGET and FRTARGET.
      - removed a protected method from dump(), added arg() to the dump
      - upgraded SORT, sortRexx to exploit IRange and stageExit, optimized
        use, and factored the sort algorithm out of sort/sortRexx.
      - multiple sort stages no longer try to share static variables...
      - the compiler just uses the stage name (not args) when naming stages
0.36  - Released July 19, 1998
      * A recompile of ALL pipes with stages using IRANGE is required.
        (CHANGE, DEAL, JOINCONT, LOCATE, LOOKUP, PICK, XLATE & ZONE)
      * pipes using NFIND, NLOCATE, STRNFIND or SORT also need to be
        recompiled
      - Added BuildIRangeExit and other methods to an updated IRange
        class.  Using 'zone range stage ...' will be faster than
        'stage range ...' when the range consists of n.c or n-c (s).
      - NFIND, NLOCATE, STRNFIND implemented via stageExit and NOT
      - Fixed bugs in, JUXTAPOSE, FIND, STRFIND, SORT, COMMAND, CHANGE
      - The compiler was not calling stageExit in the correct order when
        several calls were needed to build the stage.  (zone w1 nfind..)
0.35  - Released July 16, 1998
      - Jeff Hennick pointed out a bugglet that effected LOOKUP, ZONE and
        PICK that could occur with complex ranges, I found another bug in
        strliteral
      - Jeff Hennick updated this doc with information on IRange and DString
0.35  - Released July 15, 1998
      * A recompile of ALL pipes using ZONE, TCPCLIENT, TCPDATA, PREFIX
        and APPEND is required.
      - prefix and append can now be labeled, tcpclient and tcpdata
        now use a stage, instead of a pipe, to group data.
      - added compiler support for negitive stream numbers.  This is
        intended to be used by stageExit.  See append, prefix, tcpdata
        and tcpclient.
      - Redefined rexxArg() and stageArg() to simplify the compiler.
      - selection stages are no longer defined as final.
      - SelectInput(0) and selectOutput(0) are always called by the
        stage implementation so they can be overridden...
      - Reimplemented ZONE using stageExit, added CASEI using the same
```

```
            technique.  In theory NOT could be done the same way but, to
            avoid some recursion problems NOT is staying in the compiler.
          - StageExit modified to allow it to pass back another stage to
            call.  see ZONE, CASEI and NOT.
     0.34 - Released July 11, 1998
          - minor reportEOF(any) logic fix
          - improved command stage, threads used to process stdout and stderr.
            added zone, pad, lookup, pick, upgraded juxtapose, fixed bugs in
            specs & buffer.
          - added pad option to setIRange method
     0.33 - Released July 5, 1998
          - added rexxArg() and stageArg() methods to utils.nrx for use by the
$           compiler to query stages about what they expect their arguments to
            contain.  This allowed the compiler to be simplified.
$         - locate now handles null arguments correctly.  literals now include
            leading blanks.  Thanks for pointing out the problem René.
          - René Jansen contributed the timestamp stage.
          - logic modified to stop output() from getting an EOF when the output
            object has been peeked.  The peek status is also displayed by the
            dump() method and hense by deadlocks.
          - minor specs bug fixes (next.n and nextw.n output specs now work)
          - modified the compiler to invoke stageExit(rexx,rexx) method.  This
            allows stages to generate code and/or change the pipe topology.  See
            specs, append, prefix, change and xnop, in the stages directory.
          - modified StageError in preparation for usage changes.
          - removed the Range class - Jeff's code is better and anything that
            could be done with Range can be done using stageExit.
          - Jeff fixed bugs in change and join and added:
            fblock          joincont        notinside       outside
            inside
     0.32 - Released June 20, 1998
            Jeff updated these stages adding a few new ones too:
            abbrev          between         split           locate
            nlocate         strnfind        strfind         nfind
            find            chop
          - minor docuementation updates
          - the Range class is depreciated and will be removed.  Use the
            replacements Jeff created (see pipes\utils.nrx and stages\).
     0.31 - Released June 17, 1998
          - modified count, drop, take and deal to handle non rexx objects
            when possible
     0.31 - Released June 16, 1998
          - improved eofReport(ANY) logic to trigger when waiting on output
            and a different output stream severs.
          - factored the source for utils.class out of stages so there is
            a class to add (probably static) shared methods for all stages
          - fixed a deadlock that occured between shortStreams and exit
```

(severInput)
- Jeff Hennick updated many stages to work at CMS or near CMS levels.

append	deal	join	strfrlabel	xlate
buffer	drop	literal	strliteral	
change	fanin	locate	strtolabel	
console	fanout	split	take	
count	frlabel	strfind	tokenize	

All of Jeff's changes are GNUed. See CopyLeft.txt in the njpipes directory.

0.30 - Released May 24, 1998
- fixed logic in core classes to post all pending severs and not clear them too early either, this corrects a problem seen on Multiprocessor machines.

0.29 - www page update (docuemention) May 20
- deadlock section updated
- installation verification example corrected!

0.29 - Released May 17, 1998
- added obj2rexx stage, tolabel stage courtesy of Chuck Moore.
- enhanced change to support a single range
- Added setJITCache(Hashtable) method to pipes. This can be used to build a global object cache in programs calling pipes. The name of the Hasttable is passed to pipe/callpipe/addpipe via a cache parameter.
- Added support for reportEof options. This support is not too well tested - some good testcases are needed.

0.28 - Released May 9, 1998
- Enhanced parsing in specs (word2.1 would work, word 2.1 would not)
- Fixed COPY for a NT jit bug, fixed locate so NOT LOCATE would work, updated LITERAL not to use more than one exit(rc)
- Fixed a compiler problem that would hit multistreamed pipes using append or prefix.
- Any options not consumed by njp are passed on to nrc and java. Mainly for use from the command line, use with care in .njp files...
- Fixed shortStreams() so it works correctly when shorting streams in a stage with multiple streams.
- Tested all 8 addpipe forms and fixed runtime to work with all test cases
- modified filternjp to accept *in and *out without additional labels
- reenabled stop() in exit code...
- added gate, dam, tokenize, juxtapose and courtesy of Chuck Moore, frlabel stages

0.27 - Released May 3, 1998
- Automated the generation of in/outStream calls. For this to work the labels need to be of the form *in0: or *out0: where the '0' is replaced by the input or output stream to connect to.
- Fixed compiler/filter problems with stema

- Tighted range checking code in specs, fixed problem with delimited ranges. Specs was compiling the NetRexx EXIT command...
- Fixed a problem where output was not see that objects were consumed when using sipping pipes...
- Fixed a problem where severing an output stream did not cause the stages stacked on the node's outlist to see the sever
- Fixed a problem where the stage issuing a callpipe was not seeing the called pipe end
- Added debug option to pipes compiler
- Repaired commit and added commit levels to dump() method
- Fixed problems with callpipe servering several outputs, unstacking the saved stream was selecting it...
- Modified tcpclient and tcpdata to use a secondary thread to recieve the tcpip inputs.
- Now keep a referenced object for each pipe/stage so the JIT does not throw away its work and call/addpipes in loops work faster.
- in/outStreamState now return -1 when autocommit is enabled and the stream is unused.

0.26 - Released April 26, 1998

- Added selection methods to compiler (see getRange in section 4 and the locate stage an example#
- Added the specs stage. The compiler builds a stage to process the specs, reducing overhead.
- Added tcp/ip stages
- Fixed problems with severs using addpipe

0.25 - Optimized some stages using jinsight from www.alphaworks.ibm.com. This more than doubled the speed of some stages.
- fixed bugs in fanin, diskw
- Added netrexx filters to extract pipes, extended the functions of .njp files (multiple pipes in a file and .njp files can now contain netrexx code with pipe/callpipe/addpipe)
- fixed a timing bug in deadlock detection.
- xlate and sqlselect stages contributed by René Jansen added

0.24 - Release Feb 98

- modified the compiler so the syntax of pipes from the command line is the same as pipes from .njp files
- added the sort stage, the sortClass interface and the sortRexx example implementation
- added the timer stage

0.23 - fixed minor compiler errors (20 Dec 97)

- not stage modifier added.
- errors in this page corrected, NT install information added.
- modified diskr/diskw to use Buffered Streams.

0.22 - second public release

0.21 - enabled auto commit, stages start at a commit level of -2 and commit to a level of -1 at the first readto, peekto or output. nocommit disables the auto commit. This feature has not been

```
        completely tested (yet).
      - fixed compiler not to call netrexx if one of its pipes deadlocks
0.20  - Upgraded to May version of the NetRexx compiler (Thanks Mike!)
        this changed the compiler interface.  NetRexx from May 10 or
        later is now required.
      - nocommit added to _stages, though its a nop for now
      - modified the compiler class to use the May 10th NetRexx compiler
0.19  - initial public release (4 May 97)
```

List of Figures

List of Tables

Listings

Index

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.

ISBN 978-90-819090-3-7