

Pipelines Guide and Reference

Ed Tomlinson
Marc Remes

Jeff Hennick

René Jansen

Version 4.07-alpha of March 29, 2024

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

Publication Data

©Copyright The Rexx Language Association, 2011- 2024

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

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

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

Contents

The NetREXX Programming Series	i
1 Introduction	1
2 The Pipeline Concept	3
2.1 What is a Pipeline?	3
2.2 Stage	3
2.3 Device Driver	4
2.4 Hello world	4
2.5 Pipelines and NetREXX	5
3 Running Pipelines	7
3.1 Configuration	7
3.2 From the NetREXX Workspace (nrws) with direct execution	8
3.3 From the command line with direct execution	9
3.4 Compiled pipeline from the command line	9
3.5 Compiled pipeline from an .njp file	10
3.6 Compiled pipeline from an .njp file with additional stage definitions in NetREXX	10
4 Stage types	13
4.1 Device drivers	13
4.2 Record Selection	14
4.3 Filters	14
4.4 Other Stages	15
5 Advanced Pipelines features	17
5.1 Write your own Filters	17
5.2 Multi-Stream Pipelines	18
5.3 Pipeline Stalls	20
5.4 How to use a pipe in a NetREXX program	21
5.5 Giving commands to the operating system	25
5.6 Selecting from relational databases	25
6 The Pipes Runner	27

7	The Pipes Compiler	29
8	Built-in Stages	31
9	Differences with CMS Pipelines	101
	Index	105

The NetREXX Programming Series

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

Quick Start Guide

This guide is meant for an audience that has done some programming and wants to start quickly. It starts with a quick tour of the language, and a section on installing the NetREXX translator and how to run it. It also contains help for troubleshooting if anything in the installation does not work as designed, and states current limits and restrictions of the open source reference implementation.

Programming Guide

The Programming Guide is the one manual that at the same time teaches programming, shows lots of examples as they occur in the real world, and explains about the internals of the translator and how to interface with it.

Language Reference

Referred to as the NRL, this is meant as the formal definition for the language, documenting its syntax and semantics, and prescribing minimal functionality for language implementers.

Pipelines Guide & Reference

The Data Flow oriented companion to NetREXX, with its CMS Pipelines compatible syntax, is documented in this manual. It discusses running Pipes for NetREXX in the command shell and the Workspace, and has ample examples of defining your own stages in NetREXX.

Foreword - by Jeff Hennick

Often in programming projects, either in part or in the whole, we are faced with a collection of objects or text records where each is to be filtered and/or transformed in some way. Sometimes this is easy, many times there are special considerations to be handled.

Pipelines is specifically designed to do all the dirty work around this and by selected small already written and tested programs (called stages). NetRexx Pipelines make this quite easy. And now Pipelines, with over 150 stages, are built into NetRexx. Custom stages are easily written in NetRexx.

The concept of pipes joining small text record processing programs had its start in the early 1970s. In the 1980s, IBM greatly expanded the concept with stages that could have multiple input and output streams of records. And in the 1990s, this concept was transferred to NetRexx. NetRexx Pipelines, while handling records nicely, also adds full Java objects. NetRexx also adds some new Rexx and Java inspired stages.

Note to users coming from IBM CMS Pipelines: While many stages and pipes are and work identically, there are some inherent differences due to the underlying operating environments. While some CMS stages are not in NetRexx (APL, CP, PUNCH, etc.), NetRexx has over 30 new stages – many using concepts from NetRexx’s two parent languages, Rexx and Java.

Pipelines can read and write NetRexx variables and files. Many stages have shorter abbreviated names also to ease command line typing.

Pipes can be written on-the-fly at the command line, or made more permanent in files. Like Rexx, these can be written on a single line or in easier to read multiple lines.

Full documentation, with all the included stages (and the CMS stages not included) is in the Pipelines Guide and Reference.

Examples:

This is a classic, as would appear in a file names count.njp, It could be a single line, and run from the command line would need to be. The "--" is a stage name (alias comment) so needs to be ended with a "|". These too could be on their own lines. The count stage has other options besides words.

```
pipe (count)
  disk input.file | -- Read input file |
  count words | -- Count |
```



```
console | -- Display result
```

Here is a multi-output stage example. "<" and ">" are aliases for disk read and write. "?" is the end of a pipe. A word ending in ":" is a label. The "/"s are used to delineate the data string.

```
pipe (locrec)
  < input.file | Loc: locate /Sid/ | > selected.records ?
  Loc: | > discarded.records
```

In this one, I'll use the LITERAL and SPLIT stages to generate short contained input records to demonstrate it in action. Note: some systems will require this on a single line; some will require quote marks around everything but the pipe.

```
pipe literal aa bb;bb cc;cc dd;dd ee;ee ff;gg hh | -- input data |
  split ; | -- break the single line into many |
  between /c/ /e/ | -- make the selection |
  cons | -- see the results
```

the output is

```
cc dd
dd ee
ee ff
```

Jeff Hennick, Forth Worth, June 16th, 2023

Introduction

A Pipeline, or Hartmann Pipeline¹², is a concept that extends and improves pipes as they are known from Unix and other operating systems. The name pipe indicates an interprocess communication mechanism, as well as the programming paradigm it has introduced. Compared to Unix pipes, Hartmann Pipelines offer multiple input- and output streams, more complex pipe topologies, and a lot more.

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

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

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

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

²This page used to be called Hartmann Pipeline, but was renamed to CMS Pipelines in 2016

The Pipeline Concept

2.1 What is a Pipeline?

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

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

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

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

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

A pipeline is a sequence of two or more *stages*. The pipeline specification is processed by the *pipeline compiler*, and consists of a character string. A solid vertical bar | is used as *stage separator* (an option allows you to use a different character).³

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.

When looking at 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.

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

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 4.1 on page 13 shows the currently supported input- and output devices. The pipeline compiler connects these programs; it uses one program for each device and connects them together.

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

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

2.4 Hello world

The simplest form of a pipeline is shown below in a well known greeting :

```
pipe literal Hello, world! | console
```

This pipeline consists of two stages: *literal Hello, world!*, in plumbing terms the 'well', and *console*, the 'sink'. In this case, both stages are device drivers, *literal* pushes the following text into the pipe, and *console* shows the received text on the screen. The stages are connected by a | vertical bar, the default stage separator.

Note that the pipeline source contains characters which have special meaning on the command line in Windows, Linux and macOS. Therefore it is necessary to enclose the pipeline source within the appropriate quotes when running a pipeline from the command line. That is double quotes " on Windows, or single quotes ' on Linux and macOS. These quotes are not necessary within the *nrws* interface (see 8).

2.5 Pipelines and NetREXX

Internally, the Pipelines engine on NetREXX generates NetREXX source code from the pipeline source text. This NetREXX source code is compiled as a Java class, which is eventually run by the Java Virtual Machine.

Stages - these also are NetREXX programs compiled as Java class files - are implemented as threads.

The Java classname is generated randomly, unless a classname is given as first argument between () round brackets, e.g.

```
pipe '(hello) literal Hello, world! | console'
```

More options are available, see 27.

Note, you cannot specify options in NetREXX Workspace pipelines.

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 (pipe runner), pipc (pipe compiler) and nrc (netrexx compiler) utility programs. Aliases are preferable because some shell processors have idiosyncrasies in the treatment of script arguments. With an alias we can be sure that every NetRexx program sees its arguments the same way.

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

The bash aliases expect classpath to be exported correctly as:

```
export CLASSPATH=${NETREXX_HOME}/lib/NetRexxF.jar:.$CLASSPATH
```

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

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

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

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

Both the Windows batch files as well as the Linux shell scripts are shipped in the bin directory of the NetRexx package.

Do note that the Windows .bat files and Linux shell scripts 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 and scripts enable you to run a pipeline from the commandline, by typing:

```
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 sections which possibilities you have for running pipelines in day-to-day usage.

3.2 From the NetREXX Workspace (nrws) with direct execution

The NetREXX Workspace 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, because the NetREXX Workspace preloads the Pipelines subsystem by executing pipeline 'literal pipelines processor loaded. | console' during initialisation.

Note, the nrws.input file in your home directory allows to run more code during nrws startup.

There is no magic: we execute a pipeline which displays 'Pipe processor loaded'. This loads all necessary classes and leaves them in memory.

Then we can start specifying pipelines at the *Ready:* prompt.

```
Workspace for NetRexx 4.05 build 2,156-20230131-1212  
Copyright (c) Martin Lafaix 2000  
Copyright (c) parts RexxLA 2019,2021  
pipelines processor loaded  
Ready; pipe literal a man a plan a canal panama | change / // |  
      console                0.991 s  
amanaplanacanalpanama  
Ready;
```

Executed this way, the generated class image will not be written to disk. Note that the pipelines compiler creates NetREXX source code which is then compiled and run by the pipelines runner. All these are ephemeral within the NetREXX Workspace.

The *timing* option is great for prototyping and performance work.
Type *exit* to leave the NetREXX Workspace.

3.3 From the command line with direct execution

When using the CLI pipe command, the rest of the specification needs to be quoted in the command shells of Linux, Windows and macOS. Windows needs double quotes, zVM/CMS does not need quotes, but if they are used they need to be double quotes. Linux and macOS can use single or double quotes, in most cases.

```
$ pipe "literal a man a plan a canal panama | change / // | console"
amanaplanacanalpanama
```

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

3.4 Compiled pipeline from the command line

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.

```
$ pipc '(aplan) literal a man a plan a canal panama | change / // |
      cons'
( aplan ) literal a man a plan a canal panama | change / // | cons
$ ls aplan*
aplan.class
$ java aplan
amanaplanacanalpanama
```

This will yield a `aplan.class` classfile, which can be executed by the Java Virtual Machine.

Be sure to leave out the `.class` suffix when invoking `java`. Additional options are available in this mode:

- gen to save the generated `.nrx` file to disk, default is `-nogen`
- keep to save the from the `.nrx` generated `.java` source file, default is `-nokeep`

To specify the literal content from the command line, use the `arg()` method :

```
$ pipc '(aplan) literal arg() | change / // | reverse | cons'
( aplan ) literal arg() | change / // | reverse | cons
$ ls aplan*
aplan.class
$ java aplan a man ap
panama
```

3.5 Compiled pipeline from an .njp file

The `pipc` command accepts a given `.njp` file as argument.

When compiled from an `.njp` file, the pipe specification must not be quoted. Pipelines can be specified in so-called *Portrait Mode*, which is the standard for more complex pipelines as it is easier to read.

The given `.njp` file is compiled and runnable as a Java class file, it is not needed to specify the `.njp` file extension.

Note the difference in naming between `.njp` and `.class` file.

```
$ cat aman.njp
pipe (aplan)
  literal a man a plan a canal panama |
  change / // |
  console |
  reverse |
  console
$ pipc aman
pipe (aplan ) literal a man a plan a canal panama | change / // |
  reverse | console | reverse | console
$ ls aplan*
aplan.class
$ java aplan
amanaplanacanalpanama
amanaplanacanalpanama
```

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

When working with `.njp` files it is possible to create an additional stage in NetREXX, by coding it in the `.njp` after the pipeline specification.

The following example `length1.njp` specifies a pipeline in which one of the stages is defined in the `.njp` itself. When run, it tries to read the contents of itself and will output its lines prepended by the line length in decimal and hex.

In fact this is what the NetREXX `length1` class does. The class name must be identical as the basename of the `.njp` source file.

```
$ cat length1.njp
pipe (length2)
  < length1.njp |
  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.right(3) (l.d2x).right(2) line)
        readto()
    end
    catch StageError
    rc = rc()
    end
    exit(rc*(rc<>12))
$ pipec length1
pipe (length2 ) < length1.njp | length1      | console
$ ls length?.class
length1.class  length2.class
$ java length2
15 F pipe (length2)
17 11 < length1.njp |
17 11 length1      |
   8 8 console
   0 0
33 21 import org.netrexx.njpipes.pipes.
33 21 class length1 extends stage final
   0 0
14 E   method run()
   6 6     do
18 12         loop forever
21 15     line = rexx peekto()
16 10     l = line.length
41 29     output(l.right(3) (l.d2x).right(2) line)
   9 9     readto()
   9 9     end
20 14     catch StageError
15 F         rc = rc()
   7 7     end
21 15     exit(rc*(rc<>12))

```

Be sure to invoke the right java class, invoking length1 will have the JVM complain about a non-existing main method.

Note, when coding NetREXX stages in an .njp file, make sure the pipeline specification is separated from the NetREXX code by at least one blank line.

Stage types

Stages can be categorised in different groups : device drivers, record selection stages and filters.

Chapter 8 documents all built-in stages and differences to CMS Pipelines.

For detailed information on the built-in stages, refer to the CMS Pipelines User's Guide and Reference.

4.1 Device drivers

Pipelines for NetREXX contains the following device drivers:

TABLE 1: Device drivers

<	read from a file
>	write to a file (which is overwritten if it exists)
>>	append to a file (which is created if it does not exist)
diskr	read from a file
diskw	write to a file (which is overwritten if it exists)
diska	append to a file (which is created if it does not exist)
diskslow	read, create or append to a file
array	manipulate arrays
arraya	append to an array
arrayr	read an array
arrayw	write to an array
stem	manipulate stems
stema	append to a stem
stemr	read a stem
stemw	write to a stem
vector	manipulate vectors
vectora	append to a vector
vectorr	read elements of a vector
vectorw	write elements to a vector
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

4.2 Record Selection

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

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

TABLE 2: Record selection

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

4.3 Filters

Filters perform an operation on a single stream.

These are the main filters supported in Pipelines for NetREXX:

TABLE 3: 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	block its primary input and executes 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

4.4 Other Stages

Finally, some other stages are listed below:

TABLE 4: Other stages

query	check version and level of Pipelines for NetREXX
-- --	insert comments into a pipeline
--	insert comments into a pipeline
comment	insert comments into a pipeline

Advanced Pipelines features

In this chapter we will elaborate on more advanced Pipeline features.

5.1 Write your own Filters

So we have seen in the previous examples 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 example⁴), 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)
```

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. The stage object is 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()
```

⁴From the document CMS Pipelines Explained, by John P. Hartmann

```

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. Because the source uses pipe idioms, the regular NetREXX compiler cannot understand everything, and we need to use the pipes compiler *pipc* to compile this source. This will call the NetREXX and Java compilers at the appropriate moment. The resulting .class file needs to be on the CLASSPATH environment variable.

We can test this by building the stage and running the pipeline:

```

$ nrc bagvendt
NetRexx portable processor 4.05-GA build 2,156-20230131-1212
Copyright (c) RexxLA, 2011,2023. All rights reserved.
Parts Copyright (c) IBM Corporation, 1995,2008.
Program bagvendt.nrx
=== class bagvendt ===
  method run
    signals ThreadQ
    overrides stage.run
Compilation of 'bagvendt.nrx' successful
$ pipe 'literal a plan | bagvendt | cons'
nalp a

```

5.2 Multi-Stream Pipelines

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

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

Let us look at a simple selection like the following:

```

$ pipe "literal foo bar baz frob frobnitz frobbotzim | split | locate
      /oo/ | cons"
foo

```

The string that makes it through the *locate* selection is 'foo' - it is the only string 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 - a character string terminated by a `:` colon. We call it, in absence of any creativity, *rest*:⁵. Also, we send the selected `foo` output into a *hole* stage, where it disappears.

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

As predicted, there is no output. To get to the rest of the words which are not selected by *locate*, we connect the secondary output stream to a new pipe, using the `'?` (the default pipe-end character) and the `rest:` label like this:

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

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

```
$ pipe "literal foo bar baz frob frobnitz frobbotzim | split | rest:
      locate /oo/ | reverse | cons ? rest: | cons"
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 | cons ? rest: | locate /botzim/ | cons"
oof
frobbotzim
```

It is best practice to define and implement secondary streams when you write your own stages.

A first label connects to the first streams (in and out) of the stage. A second label connects to the secondary streams, a third to the next, etc.

As stages are threads there is no guarantee of order of execution of the additional pipelines:

```
$ cat multipipe.njp
pipe ( multipipe end ? )
  literal  eno |
  a: faninany |
  reverse |
  cons ?
  literal  owt |
```

⁵often, you will see it being called `'a:'`

```

a: ?
  literal eerht |
a: ?
  literal ruof |
a:

$ pipec multipipe
pipe (multipipe end ? ) literal eno | a: faninany | reverse | cons ?
  literal owt| a: ? literal eerht | a: ? literal ruof | a:
$ java multipipe
one
four
three
two
$ java multipipe
four
one
three
two

```

5.3 Pipeline Stalls

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

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

```

$ pipe 'literal test | a: fanin | cons | a:'
test
Deadlocked in p49b739c

Dumping p49b739c Stall 2000 Monitored by p49b739c

Flag units digit: 1=wait out, 2=wait in, 4=wait any, 8=wait commit
                  : 10=pending autocommit, 20=pending sever

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

fanin_2
Running rc=0 commit=-1 Flag=201 waits 0 args=
-> in 0 literal_1 1 test
   in 1 cons_3 0 test
-> out 0 cons_3 1 test

cons_3
Running rc=0 commit=-1 Flag=201 waits 0 args=
-> in 0 fanin_2 1 test
-> out 0 fanin_2 0 test

Dumped Pipe p49b739c Flag 60F rc=16

ThreadQ Thread[#27,Thread-1,5,njPipes]

```

```
ThreadQ Thread[#28,Thread-2,5,njPipes]
ThreadQ Thread[#29,Thread-3,5,njPipes]
compiler:RC=16
```

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 '->' arrow shows which stream is selected. From this we can see cons_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, true means the current stage has seen the value.

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

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

5.4 How to use a pipe in a NetREXX program

The following shows how to use a pipe in a NetREXX program:

```
$ cat testpipe.njp
```

```
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('This is prefixed')
        exit
$ pipe testpipe
pipe (testpipe_apipe stall 1000) stem F | sort | prefix literal arg(
    string 'avar'} | console | stem T
$ java testpipe
This is prefixed
1111
222
3333
444
55
T[1]=This is prefixed
T[2]=1111
T[3]=222
T[4]=3333
T[5]=444
T[6]=55

```

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

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

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

In `.njp` files you can use `{avar}` phrase actually just shorthand for `arg('avar')`. The following `overstem.nrx` stage 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.

```

$ cat overstem.nrx
import org.netrexx.njpipes.pipes.
class overstem 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))
$ nrc overstem
NetREXX portable processor 4.05-GA build 2,158-20230131-1734
Copyright (c) RexxLA, 2011,2023. All rights reserved.

```

Parts Copyright (c) IBM Corporation, 1995,2008.

Program overstem.nrx

```
=== class overstem ===
  method run
    signals ThreadQ
    overrides stage.run
```

Compilation of 'overstem.nrx' successful

```
$ cat overtest.njp
```

```
class overtest
  method overtest()
    S = Rexx ''

    S[0]=3
    S[1]='one'
    S[2]='two'
    S[3]='three'

    pipe (aover stall 1000)
      stem S | overstem S | console

  method main(a=String[]) static
    overtest()
    exit
```

```
$ pipc overtest.njp
```

```
pipe (overtest_aover stall 1000) stem S | overstem S | console
```

```
$ java overtest
```

```
3
one
two
three
```

The getRexx method is passed the name of a string by the pipe.

If you wish to replace a stream, this can be done using connectors. For example look at the following fragment:

```
$ cat calltest.njp
```

```
pipe (callt) 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))
```

```
$ pipc calltest.njp
```

```
pipe (callt ) literal test | calltest arg() | console
```

```
callpipe (calltest_cp1 ) gen arg(string 'a') | *o_A0:
```

```
$ java callt 10
```



```
1
2
3
4
5
6
7
8
9
10
test
```

Running the callt1 pipe with an argument of 10 passes 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 is *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.

```
$ cat addtest.njp
pipe (addt1 debug 0 ) gen 40 | addtest | console

import org.netrexx.njpipes.pipes.

class addtest extends stage final
method run() public
  do
    addpipe (locate1 debug 0) *out: | locate /0/ | *out:
    loop forever
      line = peekto()
      output('a 'line)
      readto()
    end
  catch StageError
    rc = rc()
  end
  exit(rc*(rc<>12))
$ pipc addtest
pipe (addt1 debug 0 ) gen 40 | addtest | console
addpipe (addtest_locate1 debug 0) *o_A: | locate /0/ | *o_B:
$ ls add*class
```

```
addt1.class  addtest.class  addtest_locate1.class
$ java addt1
a 10
a 20
a 30
a 40
```

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 `rex` object try using the `rex` stage modifier to attempt to convert the object.

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`. It is also used to produce better performing stages, for an example see `specs`. The compiler also queries the `rexArg()` and `stageArg()` methods. If your stage expects objects of class `Rexx` as arguments `rexArg()` 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.

5.5 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
```

Note, on Windows some commands, like `dir`, 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
```

5.6 Selecting from relational databases

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

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

The file looks like this:

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

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

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

This yields the following output:

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

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

The Pipes Runner

The *pipe* command alias starts 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 macOS⁶.

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 .nrx, .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 persistence⁷.

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

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

The following options are available:

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

⁶this is a non-exhaustive list of operating systems

⁷But specifying them will not generate an error

The Pipes Compiler

The *pipc* command alias starts the Pipes Compiler. The purpose of compiling 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 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:

```
pipc -gen -keep testpipe.njp
```

This will generate the NetREXX source as well as keep the java source for testpipe.njp.

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). 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 4.06 & CMS Pipelines V7R1 and Their Differences

How to Read Syntax Diagrams

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

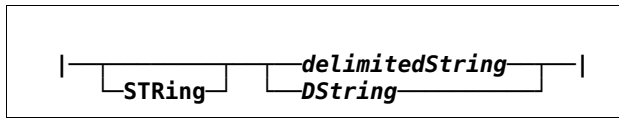
To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

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

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

Some special symbols used in the diagrams

Delimited String



Examples:

```
/abc/  
''  
xf1f2f3  
b11000001  
str xabx
```

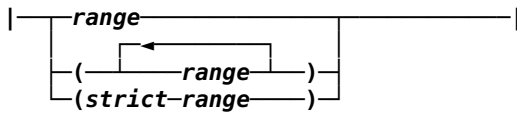
A delimited character string is written between two occurrences of a delimiter character, as a hexadecimal literal, or as a binary literal. The delimiter cannot be blank and it must not occur within the string. Two adjacent delimiter characters represent the null string. It is suggested that a special character be used as the delimiter, but this is not enforced.

A hexadecimal literal is specified by a leading H or X followed by an even number of hexadecimal digits. A binary literal is specified by a leading B followed by a string of 0 and 1; the number of binary digits must be an integral multiple of eight.

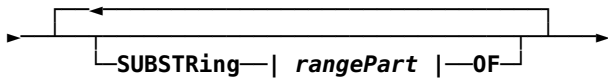
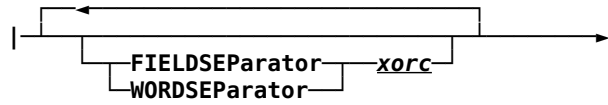
The keyword `STRING` can be used to specify that the delimited string contains a string that is terminated by delimiter characters.

Input Range

IRange:

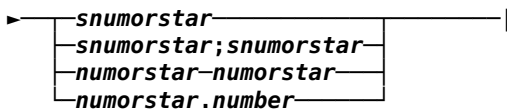
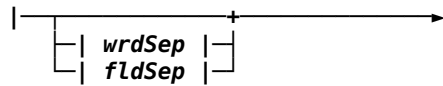


range:

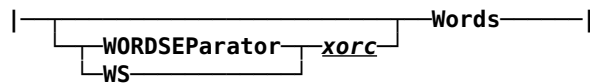


rangePart:

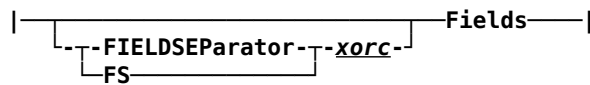
rangePart:



wrSep:



fldSep:



Examples:

```
1-*
word 5
1;-1
-18;28
field 4
```

An input range is specified as a column range, a word range, a field range.

A single column is specified by a signed number. Negative numbers are relative to the end of the record; thus, -1 is the last column of the record. A column range is specified as two signed numbers separated by a semicolon or as a range. When a semicolon is used, the first number specifies the beginning column and the second number specifies the ending column. When the beginning and end of a field are relative to the opposite ends of the record, the input field is treated as a null field if the ending column is left of the beginning column.

A word range is specified by the keyword WORDS, which can be abbreviated down to

W. Words are separated by one or more blanks. The default blank character is X'20'. Specify the keyword WORDSEPARATOR to specify a different word separator character. WORDSEPARATOR can be abbreviated down to WORDSEP; WS is a synonym.

A field range is specified by the keyword FIELDS, which can be abbreviated down to F. Fields are separated by tabulate characters. Two adjacent tabulate characters enclose a null field. (Note the difference from words.) The default horizontal tab character is X'09'. Specify the keyword FIELDSEPARATOR to specify a different field separator character. FIELDSEPARATOR can be abbreviated down to FIELDSEP; FS is a synonym.

QString

|—*qstring*—|

A quote delimited string, the quote marks may be either single or double. The string may be a empty, or a single word. If a single word, the quote marks are optional.

Examples:

- "string of words"
- "word"
- 'word'
- word
- 'She said, "Yes."'

Qword

|—*qword*—|

A space delimited word, optionally enclosed in quote marks, either single or double. Or a multi word phrase enclosed in quote marks.

Examples:

- word
- "word"
- 'word'
- "word meaning"
- 'She said, "Yes."'

Regex String

|—*regex_string*—|

Examples:

```
a
A dog
^[Aa]?\\s*dog(s)?
```

A regular expression string, or `regex_string`, defines a search pattern for strings. The search pattern can be anything from a simple character, a fixed string, or a complex expression containing special characters describing the pattern.

Using regular expressions can be very powerful. Also can be very hard to read, and nearly so to write.

Regular expressions are used in many different programming languages, and have several dialects. NetRexx Pipelines uses its underlying Java's version.

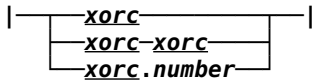
Xorc

|—*xorc*—|

Examples:

A character specified as itself (a word that is one character) or its hexadecimal representation (a word that is two characters). The blank is represented by the keyword `BLANK`, which has the synonym `SPACE`, or with its hex value, `X'20'`. The default horizontal tabulate character (`X'09'`) is represented by `TAB`.

Xrange

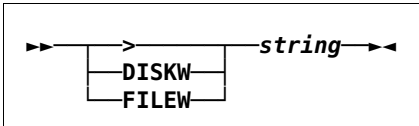
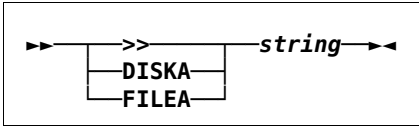


Character ranges designate the characters in the collating sequence between two specified characters; such a range is often called a hex range because the characters can be specified as `xorc`. A hex range can be written with the first and last characters separated by a hyphen ('-'), or by the first character and a count separated by a period ('.'). No blanks are allowed between the characters and the delimiters because CMS Pipelines scans for a word before scanning the word for the hex range. Hex ranges wrap from 0XFF to 0X00 when the starting character is later in the collating sequence than the ending one, or the count is larger than the number of characters from the beginning character to the end of the collating sequence.

Examples:

- `Y`
- `X-Z`
- `00-7f`
- `00.256`
- `0-00`
- `BLANK`
- `40-7f`
- `blank-7f`
- `blank.3`
- `00-blank`

Pipelines Builtin Stages

<pre>> diskw filew 3.09</pre>	<p>Replace or Create a File</p>  <ul style="list-style-type: none">• delegates to diskw.
<pre>>> diska filea 3.09</pre>	<p>Append to or Create a File</p>  <ul style="list-style-type: none">• delegates to diska.
<pre>>>mdsk</pre>	<p>Append to or Create a CMS File on a Mode</p> <ul style="list-style-type: none">• Not implemented in Netrexx Pipelines.
<pre>>>mvs</pre>	<p>Append to a Physical Sequential Data Set</p>

	<ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>>oe	Append to or Create an OpenExtensions Text File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>>sfs	Append to or Create an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>>sfsslow	Append to or Create an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>mnsk	Replace or Create a CMS File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>mvs	Rewrite a Physical Sequential Data Set or a Member of a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>oe	Replace or Create an OpenExtensions Text File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
>sfs	Replace or Create an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
< diskr filer 3.09	Read a File  <ul style="list-style-type: none"> • Implemented as in CMS; delegates to diskr.
<mnsk	Read a CMS File from a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<mys	Read a Physical Sequential Data Set or a Member of a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<oe	Read an OpenExtensions Text File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<sfs	Read an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<sfsslow	Read an SFS File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
-- comment 3.09	Comment Stage, No Operation  <ul style="list-style-type: none"> • delegates to comment. • Not in CMS Pipelines; • This is a STAGE, not a programming comment. It must have a SPACE after --. • It must have either a stageEnd or pipeEnd. • If used before a driver stage, it must have a

pipeEnd.

3277bfra

Convert a 3270 Buffer Address Between Representations

- Not implemented in Netrexx Pipelines.

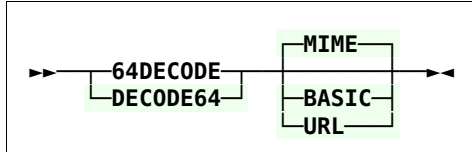
3277enc

Write the 3277 6-bit Encoding Vector

- Not implemented in Netrexx Pipelines.

64decode
decode64
3.11

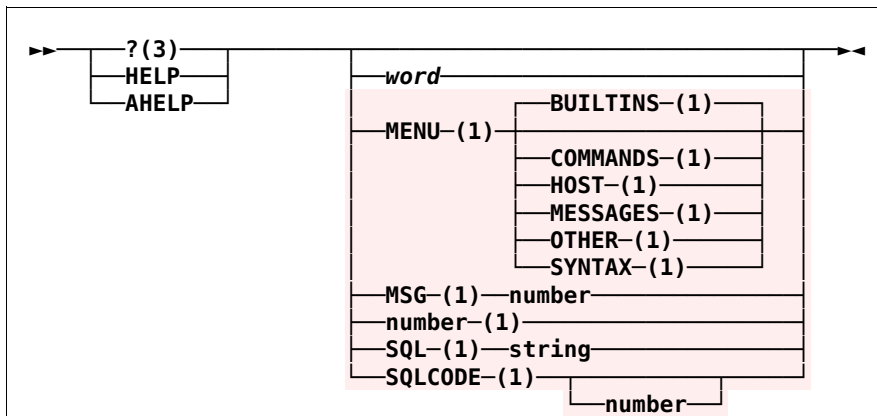
Decode Base-64 Format



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

?
ahelp
help

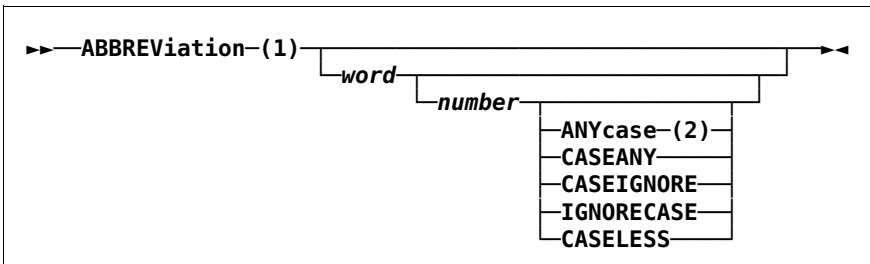
Display Help for Pipelines



- (1) CMS Pipelines only. Not yet in NetRexx Pipelines.
- (2) If primary output is connected, lines are propagated, otherwise they are sent to the console by "say."
- (3) ? is the default pipeEnd character. Here it is useful only when a different pipeEnd is defined.

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

<i>acigroup</i>	Write ACI Group for Users <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
-----------------	--

<i>addrdw</i>	Prefix Record Descriptor Word to Records <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
---------------	---

<i>adrspace</i>	Manage Address Spaces <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
-----------------	--

<i>aftfst</i>	Write Information about Open Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
---------------	---

<i>aggrc</i>	Compute Aggregate Return Code <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> >> AGGRC << </div>
--------------	--

<i>ahelp</i> <i>help</i> ?	Display Help for Pipelines <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre> graph LR AHELP[">> AHELP <<"] --- HELP["HELP"] AHELP --- Q["?-(3)"] AHELP --- word["word"] word --- MENU["MENU-(1)"] MENU --- BUILTINS["BUILTINS-(1)"] BUILTINS --- COMMANDS["COMMANDS-(1)"] BUILTINS --- HOST["HOST-(1)"] BUILTINS --- MESSAGES["MESSAGES-(1)"] BUILTINS --- OTHER["OTHER-(1)"] BUILTINS --- SYNTAX["SYNTAX-(1)"] word --- MSG["MSG-(1) number"] word --- NUM["number-(1)"] word --- SQL["SQL-(1) string"] word --- SQLCODE["SQLCODE-(1) number"] </pre> </div> <p>(1) CMS Pipelines only. Not yet in NetRexx Pipelines. (2) If primary output is connected, lines are propagated, otherwise they are sent to the console by "say." (3) ? is the default pipeEnd character. Here it is useful only when a different pipeEnd is defined.</p>
----------------------------------	--

<i>all</i> 4.06	Select Lines Containing Strings (or Not) <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> >> ALL expression << <table style="margin-left: 20px;"> <tr><td>%debug</td><td>(6)</td></tr> <tr><td>%dump</td><td>(7)</td></tr> <tr><td>%see</td><td>(8)</td></tr> </table> </div>	%debug	(6)	%dump	(7)	%see	(8)
%debug	(6)						
%dump	(7)						
%see	(8)						

- Notes:
- (1) "expression" consists of one or more delimited strings separated by logical ANDs, ORs, and NOTs, and grouped, if needed, by parentheses.
 - (2) "&" is used for AND.

- (3) Since "|" is the default stage separator, "!" may be used for OR.
- (4) Since NetRexx uses "(" and ")" for options -- which are not used in the ALL stage -- "[" and "]" must be used for parentheses.
- (5) CMS Pipelines, having originated on 3270 terminals, uses "-" for NOT. This symbol is not readily typed on terminals running NetRexx Pipelines, so as alternatives, "\\ ", used by NetRexx, (it needs to be doubled to "escape" it) or "^", used by KEX, NOT symbols may be used as alternatives.
- (6) %debug (must be lowercase) NetRexx Pipelines writes the logic line to the file ALL.DEBUG in the current directory. Windows may make it all.debug . CMS Pipelines writes the constructed pipeline (of LOCATE and NLOCATE stages) to ALL DEBUG A.
- (7) %dump (must be lowercase) - writes to the primary output stream as the first record. NetRexx Pipelines writes the logic line. CMS Pipelines writes constructed pipeline.
- (8) %see (must be lowercase) - NetRexx Pipelines Only. Writes the logic line to the standard output (terminal).
- CMS Pipelines uses its own logic order. NetRexx Pipelines uses regular NetRexx logic.

Examples:

- `literal NetRexx is Good,NetRexx is Great,NetRexx is Fantastic |`
`split , |`
`all /a/ |`
`cons`

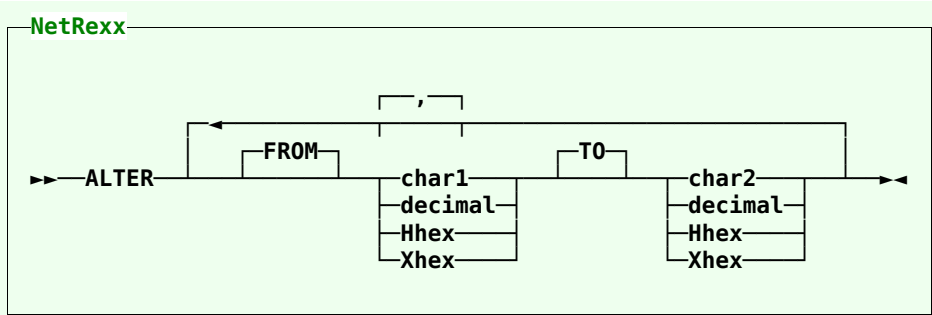
`▶NetRexx is Great`
`▶NetRexx is Fantastic`
- `literal NetRexx is Good,NetRexx is Great,NetRexx is Fantastic |`
`split , |`
`all / G/ & [/oo/ ! /F/] |`
`cons`

`▶NetRexx is Good`
- `literal NetRexx is Good,NetRexx is Great,NetRexx is Fantastic |`
`split , |`
`all /R/ & [/oo/ ! /F/] |`
`cons`

`▶NetRexx is Good`
`▶NetRexx is Fantastic`

alter

Change the contents of records, from one character to another



- An variation on the theme of Xedit's ALTER.
- There are pairs of char1s and char2s, optionally separated by commas.
- For each pair of char1 and char2, this changes ALL char1s to char2, like Xedit's 4th and 5th parameters were 1 and 1.
- The chars can be single characters, 2 or 3 digit decimal numerical representations, or beginning with H, h, X, or x hexadecimal representations.
- Also see [TRANSLATE / XLATE](#).
- Not in CMS Pipelines.

alserv

Manage the Virtual Machine's Access List

- Not implemented in Netrex Pipelines.

apldecode

Process Graphic Escape Sequences, Old APL language

- Not implemented in Netrex Pipelines.

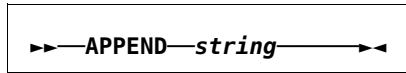
aplencode

Generate Graphic Escape Sequences, Old APL language

- Not implemented in Netrex 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 Netrex Pipelines.

asmcont

Join Multiline Assembler Statements

- Not implemented in Netrex Pipelines.

asmfind

Select Statements from an Assembler File as XEDIT Find

- Not implemented in Netrex Pipelines.

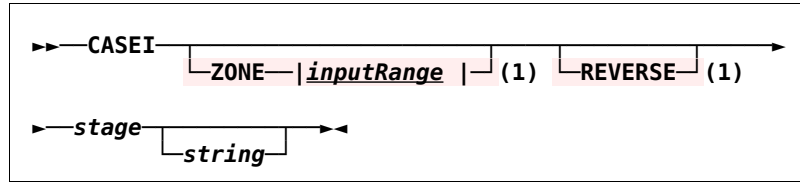
asmnfind

Select Statements from an Assembler File as XEDIT NFind

	<ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>asmxpnd</i>	Expand Joined Assembler Statements <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>beat</i>	Mark when Records Do not Arrive within Interval <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>between</i> 3.09	Select Records Between Labels <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p> </p> </div>
<i>bfs</i> <i>hfs</i>	Read or Append File in the Hierarchical File System <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>bfsdirectory</i> <i>bfsdir</i> <i>hfsdirectory</i> <i>hfsdir</i>	Read Contents of a Directory in a Hierarchical File System <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>bfsquery</i> <i>bfsq</i> <i>hfsquery</i> <i>hfsq</i>	Write Information Obtained from OpenExtensions into the Pipeline <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>bfsreplace</i> <i>bfsrep</i> <i>hfsreplace</i> <i>hfsrep</i>	Replace the Contents of a File in the Hierarchical File System <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>bfsstate</i> <i>bfsstat</i> <i>hfsstate</i> <i>hfsstat</i>	Obtain Information about Files in the Hierarchical File System <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>bfsxecute</i> <i>bfsx</i> <i>hfsxecute</i> <i>hfsx</i>	Issue OpenExtensions Requests <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>block</i>	Block to an External Format <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>browse</i> <i>brw</i>	Display Data on a 3270 Terminal <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>buffer</i>	Buffer Records <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p> </p> </div>

casei

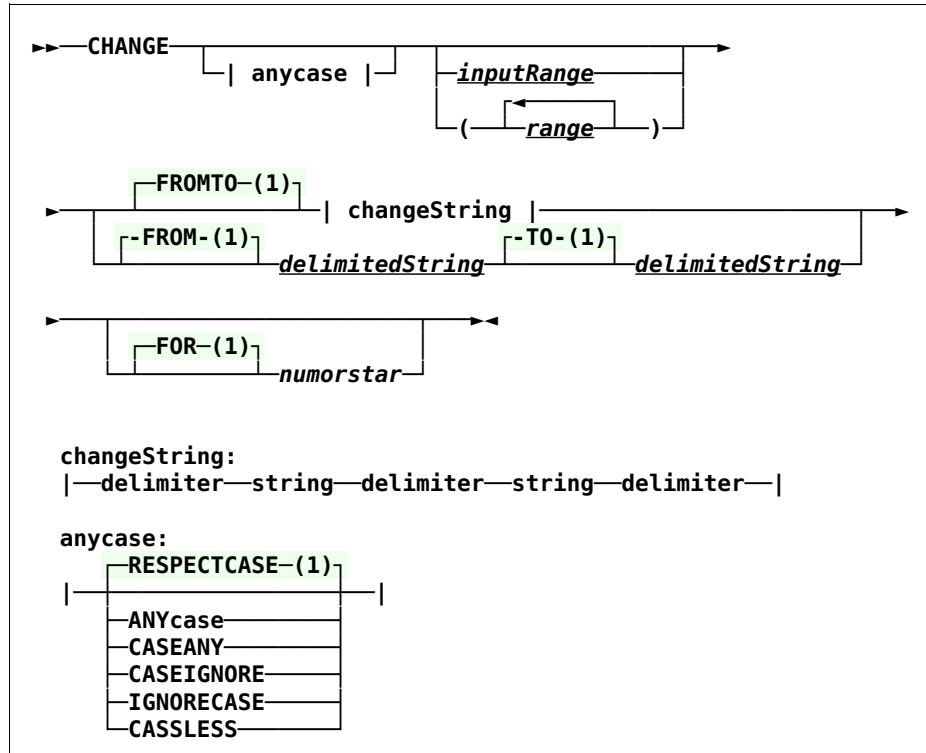
Run Selection Stage in Case Insensitive Manner



- (1) CMS Pipelines only.

change

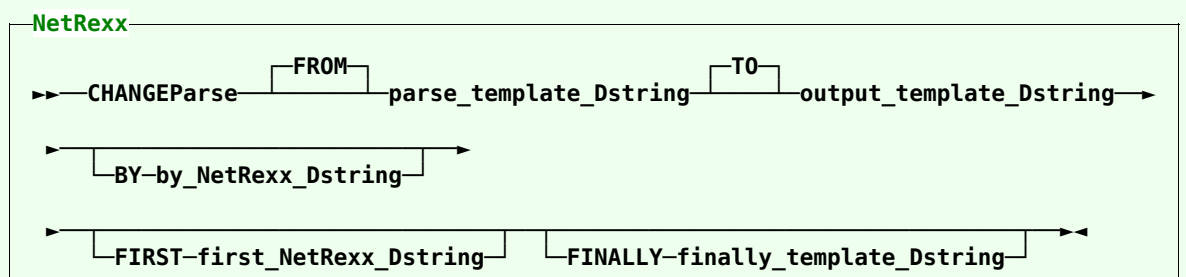
Substitute Contents of Records



- (1) NetRexx Pipelines only.

changeparse
changepars
changepar
changepa
changep
3.11

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



- Records are parsed via the parse_template_delimited_string.
- Variables are named \$n, where n is 1 to 9.
- The by_NetRexx_delimited_string is interpreted. This is 0 or more semicolon separated NetRexx statements, probably using the \$n variables, which can have the value altered.
- Other variables may be used, and are persistent while the stage is active, so can be

- used as accumulators.
- The values of the variables are put into the output_template_delimited_string replacing \$n.
- For a literal \$n that won't be changed, use \$\$n.
- A first_NetRexx_delimited_string, if present, is interpreted before reading any record from the primary input steam. This is 0 or more semicolon separated NetRexx statements, probably using the \$n variables. Any variables used in the by_NetRexx_delimited string must be defined here.
- A finally_template_delimited_string, if present, is written as a final output record after the primary input stream is finished, using the \$n's.
- Any keyword phrases must, in any order, follow any non-keyworded FROM & TO phrases.
- This is NetRexx Pipelines only, not in CMS.

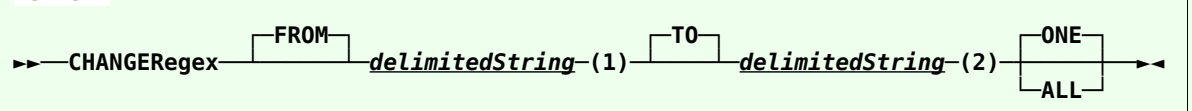
Examples:

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

changeregex
changerege
changereg
changere
changer
3.09

Substitute Contents of Records using Java Regular Expressions

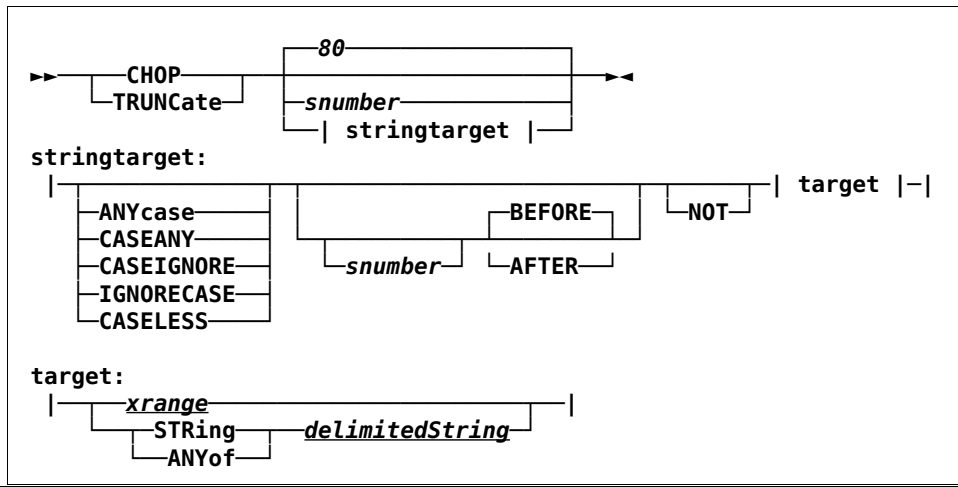
NetRexx



- 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, FROM, delimitedString is a Java RegEx expression for what is to be replaced.
- (2) Second, TO, delimitedString is the replacement string. It may contain elements from the first one.
- This is NetRexx Pipelines only, not in CMS.

chop
truncate
truncat
trunca
trunc

Truncate the Record



cipher **Encrypt and Decrypt Using a Block Cipher**

- Not implemented in Netrexx Pipelines.

ckddebblock **Deblock Track Data Record**

- Not implemented in Netrexx Pipelines.

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

The diagram for the `cmd` command shows the following structure:

- `CMD` and `COMMAND` are options that lead to a box containing `string`.

- 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 **3.11** **Collate Streams**

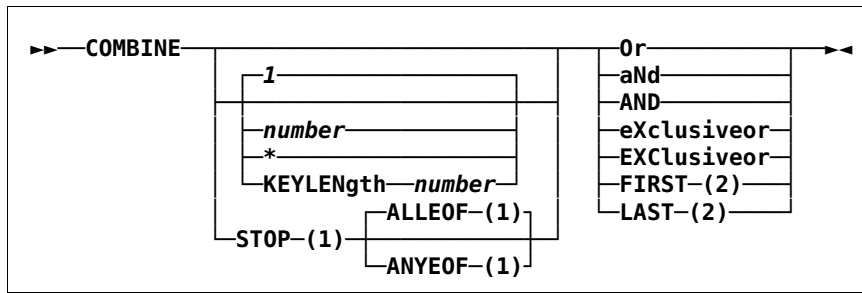
The diagram for the `collate` command shows the following structure:

- `COLLATE` is the main command.
- `STOP ANYOF` and `NOPAD` are options that lead to a box containing `PAD-xorc` and `anycase`.
- `inputRange` is an option that leads to a box containing `MASTER-DETAIL`, `MASTER-DETAIL`, and `MASTER`.
- `SEPARATOR-delimitedString-(1)(2)` is an option.
- `anycase:` is a sub-section containing:
 - `RESPECTCASE-(1)`.
 - A list of case options: `ANYcase`, `CASEANY`, `CASEIGNORE`, `IGNORECASE`, `CASSLESS`.

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

combine

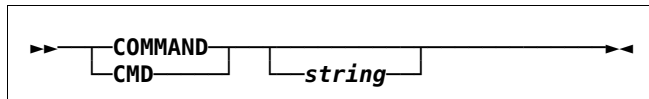
Combine Data from a Run of Records



- (1) Only for use with secondary input streams. Only options from this column usable with any secondary input 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

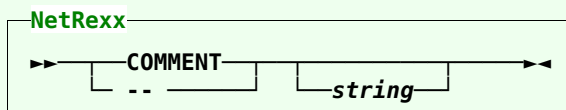
Issue OS Commands, Write Response to Pipeline



- 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

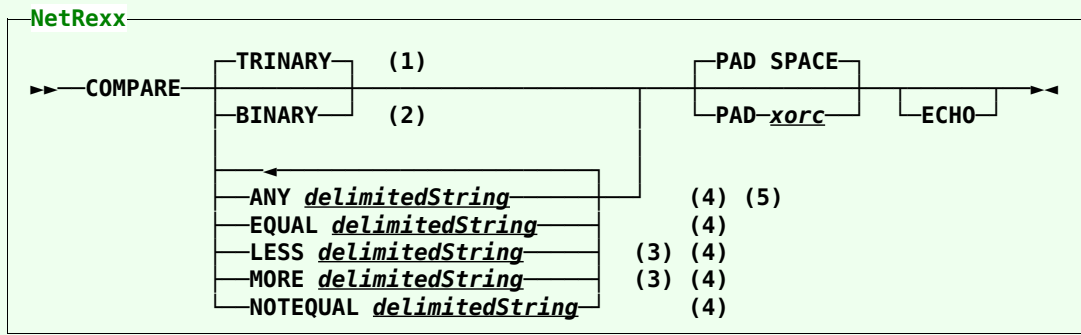
Comment stage



- 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



- (1) -1 = Primary is shorter/less, 0 = equal, 1 = Secondary is shorter/less
- (2) 0 = equal, 1 = not equal
- (3) Primary is LESS/shorter (or MORE/longer) than secondary
- (4) *DStrings* can use any of the following escapes (or the lowercase) for the unequal situation:
 - \C (count) for the record number,
 - \B (byte) for column number
 - \P (primary) for the primary stream record
 - \S (secondary) for the secondary stream record
 - \L (Least) for the stream number that is shorter, -1 if equal
 - \M (Most) for the stream number that is longer, -1 if equal
- (5) Equal or not, this *DString* precedes any of the others.
- (6) This is NetRexx Pipelines only, not included in CMS
- (7) In reporting \P & \S, control characters, except new line, \n, are transliterated to [blob, 219.d2c()]
- (8) Without ECHO, this stops and reports at first non-compare. With ECHO, each primary input is reported; after first 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

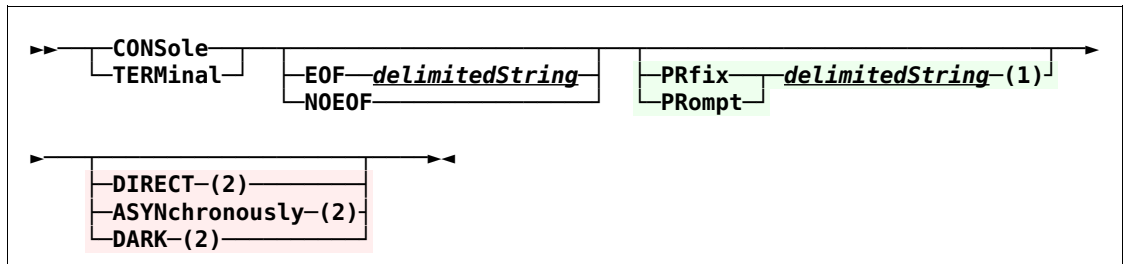
Set and Query CMS Pipelines Configuration Variables

- Not implemented in Netrex pipelines.

console
consol
conso
cons
cons
terminal
termina
termin

Read or Write the Terminal in Line Mode

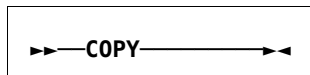
termi
term
3.11



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

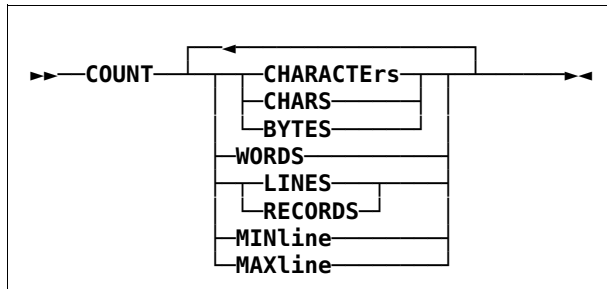
copy

Copy Records, Allowing for a One Record Delay



count

Count Lines, Blank-delimited Words, and Bytes



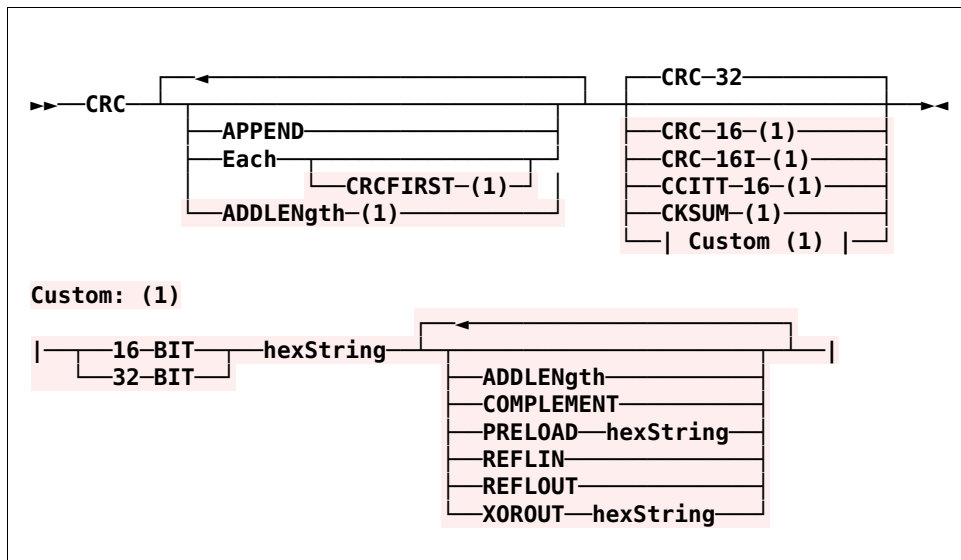
cp

Issue CP Commands, Write Response to Pipeline

- Not implemented in Netrex Pipelines.

crc
4.06

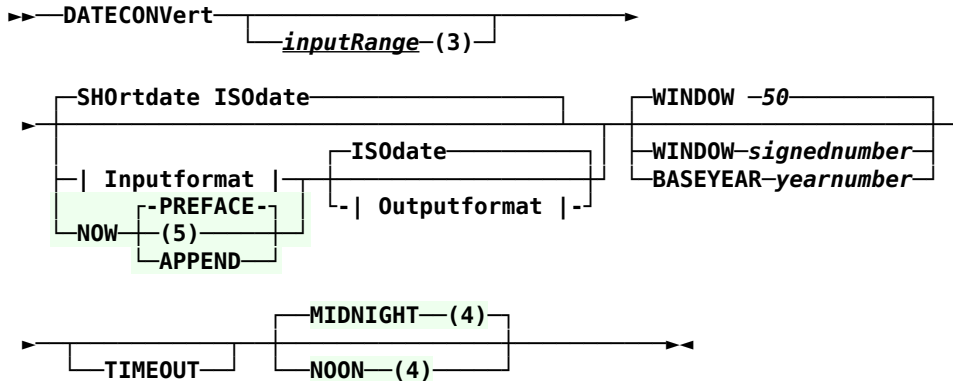
Compute Cyclic Redundancy Code



- (1) Not implemented in Netrex Pipelines.
- (2) CRC stage uses secondary output, if connected.

dateconvert
dateconver
dateconve
dateconv
3.09

Convert Date Formats



Inputformat,	Outputformat:
SHORTdate } USA_SHORT } REXX_DATE_U }	mm/dd/yy hh:mm:ss.uuuuuu
FULLdate } USA }	mm/dd/yyyyyy hh:mm:ss.uuuuuu
ISO_SHORT } ISOdate } DB2_SHORT } DB2 }	yy-mm-dd hh:mm:ss.uuuuuu yyyyyy-mm-dd hh:mm:ss.uuuuuu yy-mm-dd-hh.mm.ss.uuuuuu yyyyyy-mm-dd-hh.mm.ss.uuuuuu
VMDATE (2) NORMAL }	dd mmm yyyy yy hh:mm:ss.uuuuuu
CSL_SHORT } REXX_DATE_0 }	yy/mm/dd hh:mm:ss.uuuuuu
CSL } PIPE_SHORT } PIPE }	yyyyyy/mm/dd hh:mm:ss.uuuuuu yymmddhhmmssuuuuuu yyyyymmddhhmmssuuuuuu
REXX_DATE_S } EURSHORT } EUR }	dd.mm.yy hh:mm:ss.uuuuuu dd.mm.yyyyyy hh:mm:ss.uuuuuu
JULIAN_SHORT } JULIAN }	yy.ddd hh:mm:ss.uuuuuu yyyyyy.ddd hh:mm:ss.uuuuuu
TOD_ABSOLUTE } (2) TODABS } (2) SCIENTIFIC_ABSOLUTE } (2) SCIABS } (2) POSIX } TOD_RELATIVE } (2) TODREL } (2) SCIENTIFIC_RELATIVE } (2) SCIREL } (2) MET (2)	ssssssss

The following can be REXX_DATE_x, REXXx, or Rx

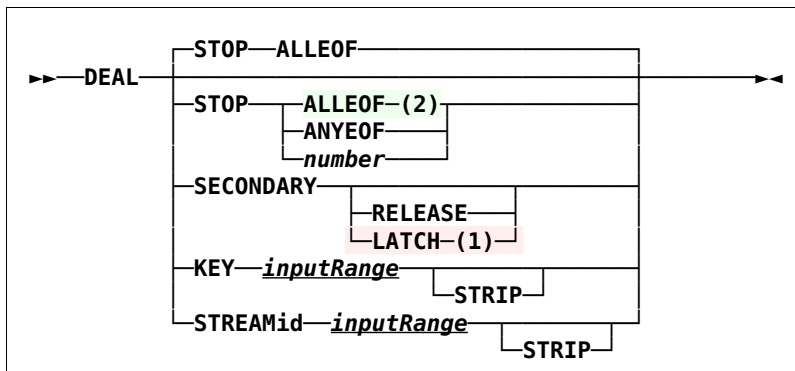
REXX_DATE_B (2)	
REXX_DATE_C (2)	
REXX_DATE_D	ddd hh:mm:ss.uuuuuu
REXX_DATE_E	dd/mm/yy hh:mm:ss.uuuuuu
REXX_DATE_E_LONG	dd/mm/yyyyyy hh:mm:ss.uuuuuu
REXX_DATE_J	yyddd hh:mm:ss.uuuuuu
REXX_DATE_J_LONG	yyyyddd hh:mm:ss.uuuuuu
REXX_DATE_M	mmmmmmmm (output only)
REXX_DATE_N_SHORT	dd mmm yy hh:mm:ss.uuuuuu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

- (1): SPACE is optional here.
- (2) Not implemented in NetRexx Pipelines at this time; mainly mainframe useful only.

- (3): NetRexx Pipelines uses IRange which gives a superset of range options.
- (4): NetRexx Pipelines only. What time to assume if blank time on input.
- (5): NetRexx Pipelines only.
 - Use current local date time.
 - Any Inputrange is ignored.
 - Any output format can be used.
 - PREFACE Write the date record before passing the input to the output.
 - APPEND Write the output record after passing the input to the output.
- Bad date records out on secondary. Or error if not connected.
- NetRexx Pipelines is more forgiving on input format:
 - LONG, _SHORT, and normal can all read either 2- or 4-digit years and process per the format;
 - leading zeros/blanks can be ignored except where there is no punctuation.

deal

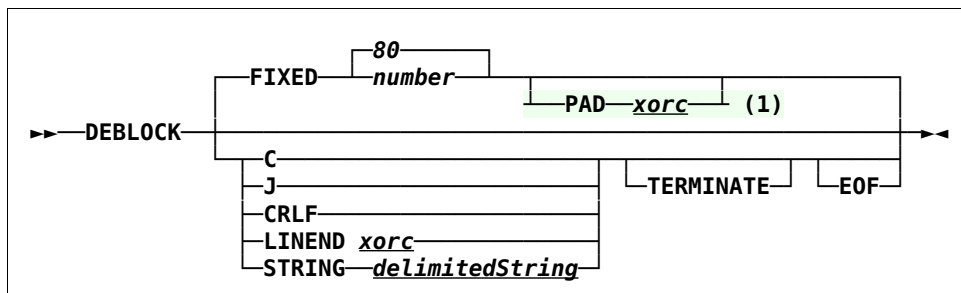
Pass Input Records to Output Streams Round Robin



- (1) Not yet in NetRexx Pipelines
- (2) Not CMS
- Since Java dispatches the stage threads, DEAL may not see a sever immediately, as the severing thread can get multitasked. This can make options like 'ANYEOF' work in unexpected ways.

deblock

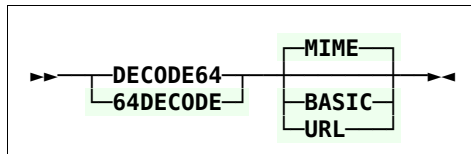
Deblock External Data Formats



- CMS has many more mainframe centric formats that NetRexx Pipelines does not process.
- (1) Not CMS Pipelines

decode64
64decode
3.11

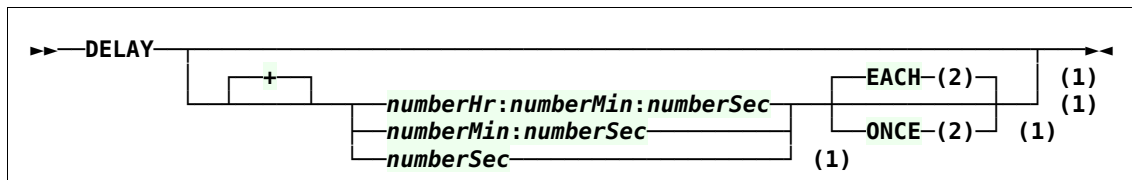
Decode Base-64 Format



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

delay
4.06

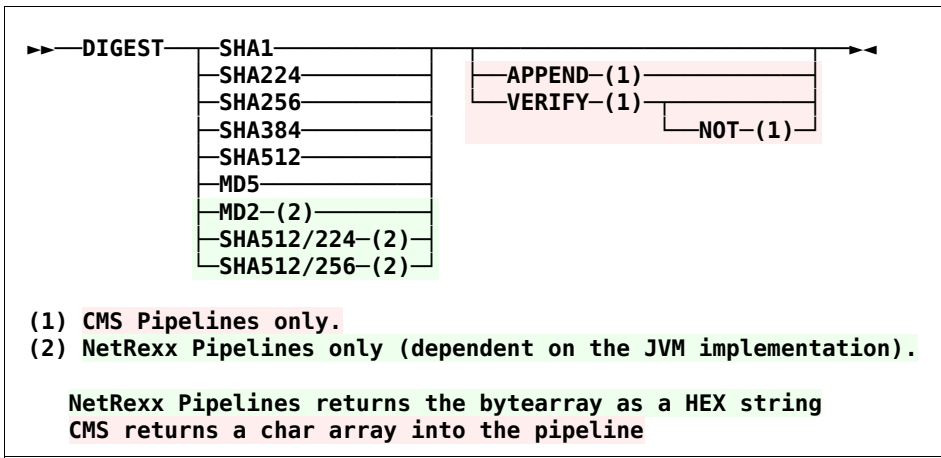
Suspend Stream



- (1) Arguments are NetRexx Pipelines only, not CMS. CMS (and NetRexx when there is no argument) reads delays as the first word of each record.
When arguments are present, they follow the CMS conventions for the delay time in records. The + indicates a duration, no + means time of day. The objects do NOT have the delay as the first word.
Clock hours are 24h, so 2pm is 14, and are for the next 24 hours if before "now."
Seconds can have decimal point and milliseconds.
In relative times, the number of seconds and

- minutes are not limited to 60; so 120 seconds is the same as 2 minutes.
- (2) Used only for "relative time." EACH, the default, delays before each object; ONCE delays only the first object.
- Uses Java's Thread.sleep() method and may not be exact in fractional seconds.

<i>devinfo</i>	Write Device Information <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>dfsort</i>	Interface to DFSORT/CMS <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>diage4</i>	Submit Diagnose E4 Requests <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>dict</i> <i>hash</i>	Read or Write a Dictionary <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>NetRexx</p> </div> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>dicta</i> <i>hasa</i>	Write a Dictionary <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>NetRexx</p> </div> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>dictr</i> <i>hasr</i>	Read a Dictionary <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>NetRexx</p> </div> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>dictw</i> <i>hasw</i>	Write a Dictionary <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>NetRexx</p> </div> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>digest</i>	Compute a Message Digest



disk
file

Read a File

►► — DISK — string —►►
 |
 — FILE —

- As in CMS, equivalent to disk (Pipes for NetRexx Only) or <.

diska
filea
>>

Append to or Create a File

►► — DISKA — string —►►
 |
 — FILEA —
 >>

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
filer
<

Read a File

►► — DISKR — string —►►
 |
 — FILER —
 <

- As in CMS, equivalent to disk (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

►► — DISKSLow — string —►►
 |
 — FILESLow —

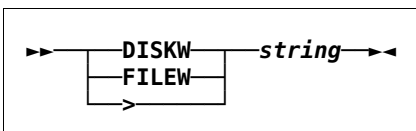
diskupdate
fileupdate

Replace Records in a File

- Not implemented in Netrexx Pipelines.

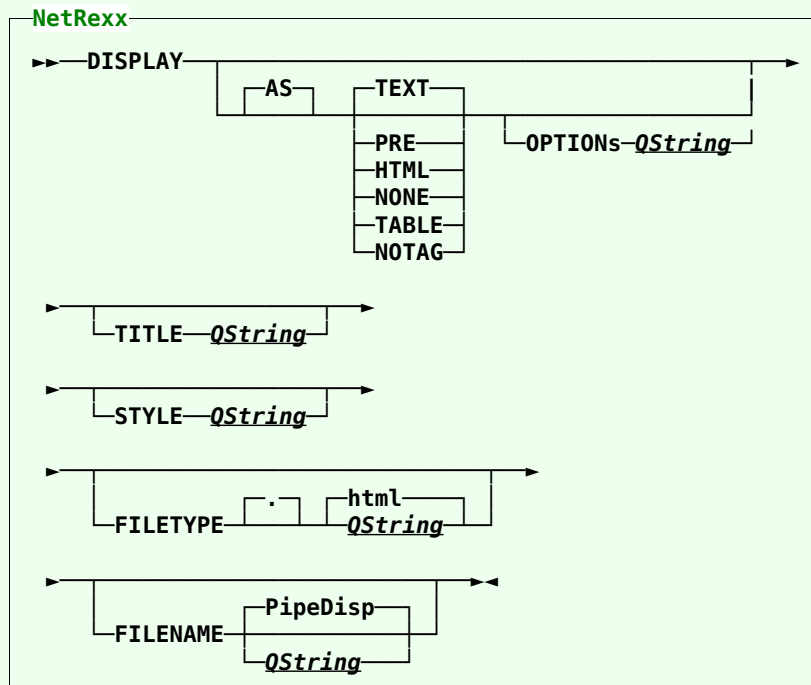
diskw
filew
>

Replace or Create a File



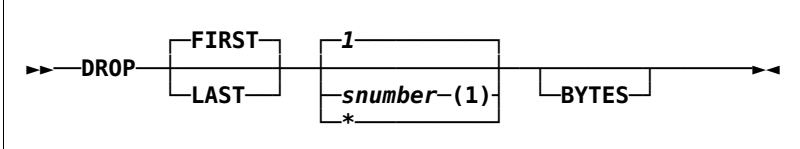
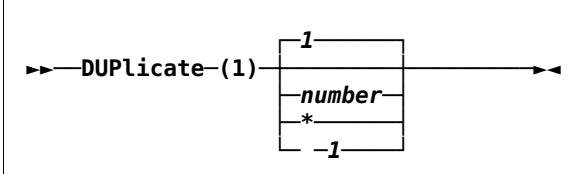
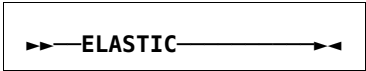
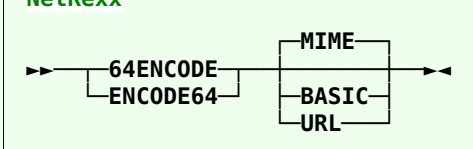
display
3.11

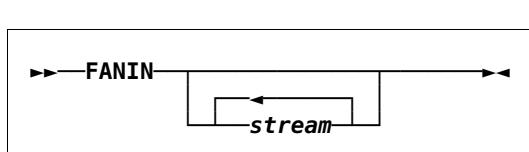
Output to Web Browser



- DISPLAY works similar to and as a replacement for CONSOLE for output. But instead of going to the terminal window, it goes to a HTML file browser tab. This allows for HTML+CSS tags to control fonts, colors, and layout.
- To work, these are required outside Pipelines and NetRexx:
 - A working HTML browser program
 - The operating system to associate the filetype "html" with the browser, so the Pipelines stage "COMMAND PipeDisp.html" does call the browser and display the file.
 - The system have a Temp directory, known to Java.
- The DISPLAY stage overwrites the named file, by default PipeDisp.html, in the system Temp directory, then calls the COMMAND stage to display it. The file is not erased automatically by this stage.
- Each DISPLAY stage invocation opens a new browser tab, which remains open.
- The AS option causes the data to be surrounded by html tags.
 - The default TEXT or PRE puts on <pre> and </pre>. Most browsers use:
 - Fixed width font
 - Display all the white spaces: line feeds and multiple spaces
 - HTML uses <html> and </html>. Most browsers use:
 - Variable width font
 - Consolidate strings of white space into a single space
 - All the HTML tags
 - TABLE uses <table> and </table>

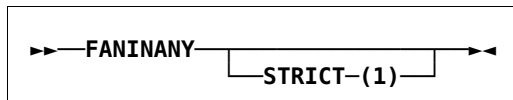
- Expects the data records to begin with `<tr>`
`<td>` (or `<tr><th>`)
 - NOTAG uses `<pre>` & `</pre>`, but first converts all `&` characters to the entity `&`; and `<` characters to `<`; so HTML tags are not processed.
 - NONE uses no extra tags. Most browsers use:
 - HTML display
- OPTIONS QString is included in the opening tag for the AS option. This could be CLASS, STYLE, or other options.
- TITLE QString adds `<title>`delimitedString`</title>` to the beginning of the output. This should show as the title in the browser's tab.
Note: This officially should go into a HEAD section; here it won't be there. Most modern browsers will honor it anyplace in the file. If it is not honored as a tag, QString will be the top line of the display.
- STYLE QString adds `<link rel="stylesheet" href="QString">` to the beginning of the output. This should include and use the named stylesheet. The name may have relative path names, or be an absolute file name. If there are spaces, enclose it in quotes.
Note: This officially should go into a HEAD section; here it won't be there. Most modern browsers will honor it anyplace in the file. If it is not honored as a tag, it will not show -- except in the NOTAG option. The file itself is copied from its stated location into the system Temp directory, overwriting any existing file. This file is not erased automatically by this stage.
QString: It is optional to enclose the name in quotes, but quotes are required if the name includes spaces.
- FILETYPE may be used to change the default "html". This permits use of other types that MAY be preprocessed if the system, external to Pipelines, is set up to recognize it, for example, "JSP" or "PHP". A "dot" is optional; only one will be used.
Note: filetypes other than .html may be handled by the system by some program other than the browser.
QString: It is optional to enclose the type in quotes.
- FILENAME may be used to write and display another file. It may include a path designation, either absolute or relative. A relative path is based on the working directory. If no path is specified in the name, the system Temp directory, as determined by Java, is used.
QString: It is optional to enclose the name in quotes, but quotes are required if the name includes spaces.
- Records from the primary input stream are also put out on the primary output stream unchanged, if it is connected.

<p><i>drop</i></p>	<p>Discard Records from the Beginning or the End of the File</p>  <ul style="list-style-type: none"> (1) CMS: must be positive. NetRexx Pipelines: negative reverses FIRST/LAST, so DROP FIRST -3 is the same as DROP LAST 3.
<p><i>duplicate</i> <i>duplicat</i> <i>duplica</i> <i>duplic</i> <i>dupli</i> <i>dupl</i> <i>dup</i></p>	<p>Copy Records</p>  <ul style="list-style-type: none"> (1) CMS is DUPLICAT due to 8-character name limitation
<p><i>elastic</i></p>	<p>Buffer Sufficient Records to Prevent Stall</p> 
<p><i>encode64</i> <i>64encode</i> <i>3.11</i></p>	<p>Encode to Base-64 Format</p> <p>NetRexx</p>  <ul style="list-style-type: none"> NOTE: CMS is only 64DECODE, and does not have the options; it does MIME. BASIC - Output is mapped to a set of characters lying in A-Za-z0-9+/. The encoder does not add any line feed in output, and the decoder rejects any character other than A-Za-z0-9+/. URL - Output is mapped to set of characters lying in A-Za-z0-9+_. Output is URL and filename safe. MIME - Output is mapped to MIME friendly format. Output is represented in lines of no more than 76 characters each, and uses a carriage return 'r' followed by a linefeed 'n' as the line separator. No line separator is present to the end of the encoded output. 3.11: New to NetRexx. Add MIME, BASIC, & URL options.
<p><i>eofback</i></p>	<p>Run an Output Device Driver and Propagate End-of-File Backwards</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>escape</i></p>	<p>Insert Escape Characters in the Record</p> <ul style="list-style-type: none"> Not implemented in Netrex Pipelines.
<p><i>fanin</i></p>	<p>Concatenate Streams</p>



faninany

Copy Records from Whichever Input Stream Has One



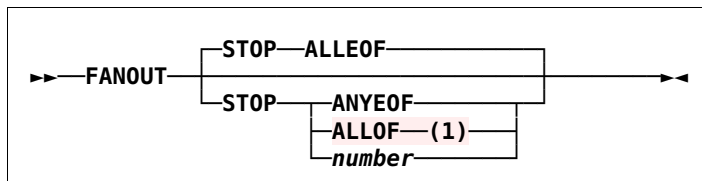
- (1) CMS only.

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.

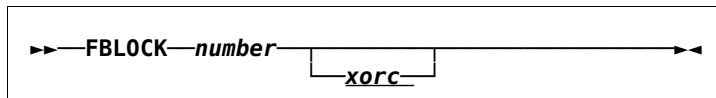
fbawrite

Write Blocks to a Fixed Block Architecture Drive

- Not implemented in Netrexx Pipelines.

fblock

Block Data, Spanning Input Records



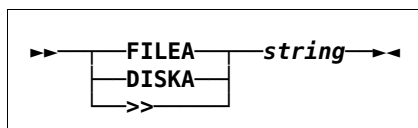
*file
disk*

Read or Write a File



*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.

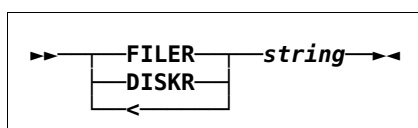
*filefast
diskfast*

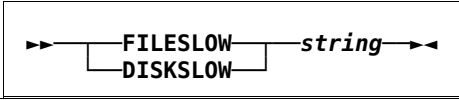
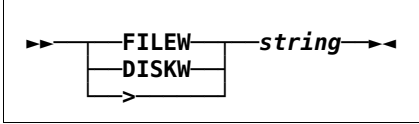
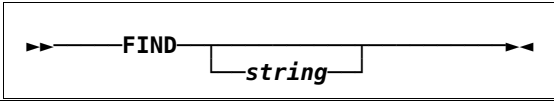
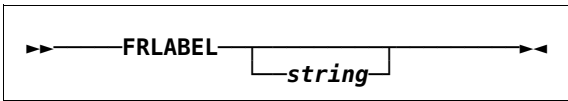
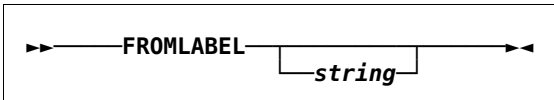
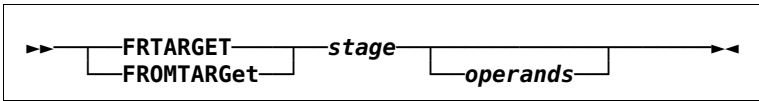
Read or write a CMS file

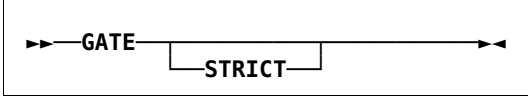
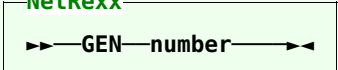
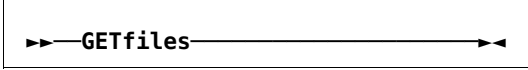

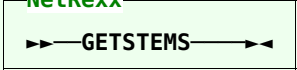
- Not implemented in Netrexx Pipelines.

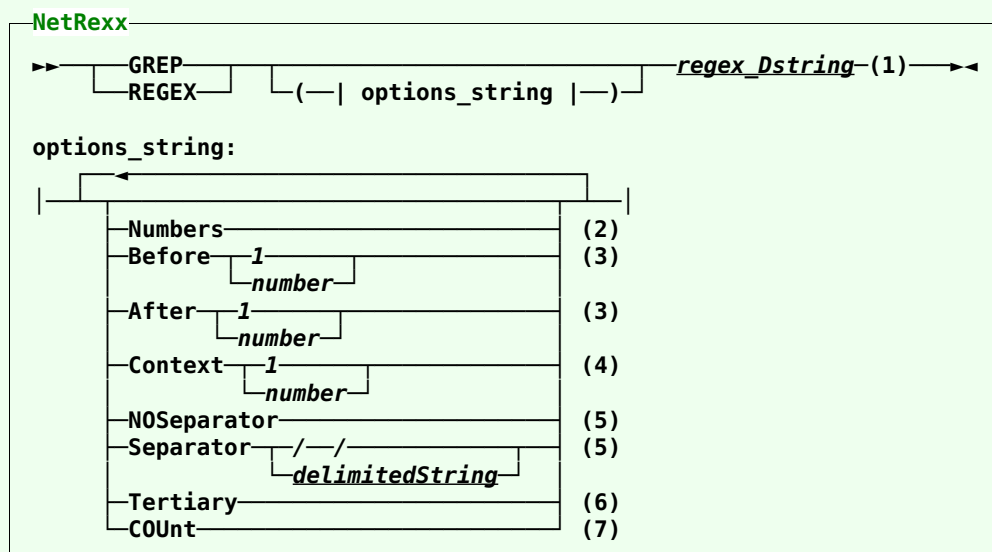
*filer
file
disk
diskr
<*

Read a File



<i>filerandom</i> <i>diskrandom</i>	Read specific records from a CMS file
<i>fileslow</i> <i>diskslow</i>	Read, Create, or Append to a File 
<i>filetoken</i>	Read or Write an SFS File That is Already Open <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fileupdate</i> <i>diskupdate</i>	Change records in a CMS file
<i>filew</i> <i>diskw</i> >	Replace or Create a File 
<i>fillup</i>	Pass Records To Output Streams <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>filterpack</i>	Manage Filter Packages <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>find</i>	Select Lines by XEDIT Find Logic 
<i>fitting</i>	Source or Sink for Copipe Data <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fntfst</i>	Format a File Status Table (FST) Entry <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>frlabel</i> <i>fromlabel</i>	Select Records from the First One with Leading String 
<i>fromlabel</i> <i>frlabel</i>	Select Records from the First One with Leading String 
<i>frtarget</i>	Select Records from the First One Selected by Argument Stage 
<i>fullscreen</i> <i>fullscree</i> <i>fullscre</i> <i>fullscr</i>	Full screen 3270 Write and Read to the Console or Dialed/Attached Screen <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fullscrq</i>	Write 3270 Device Characteristics <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>fullscrq</i>	Write 3270 Device Characteristics <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.

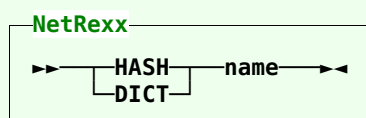
<i>fullscrs</i>	<p>Format 3270 Device Characteristics</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>gate</i>	<p>Pass Records Until Stopped</p> 
<i>gather</i>	<p>Copy Records From Input Streams</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>gen</i>	<p>Generate a Sequence of Numbers Starting with 1</p> <p>NetRexx</p>  <ul style="list-style-type: none"> • Not implemented in CMS Pipelines.
<i>getfiles</i> <i>getfiles</i> <i>getfile</i> <i>getfil</i> <i>getfi</i> <i>getf</i> <i>get</i>	<p>Read Files</p> 
<i>getovers</i>	<p>Write the Contents of Objects</p> <p>NetRexx</p>  <ul style="list-style-type: none"> • Input stream 0 should contain rexx objects. The getovers stage will output the index and contents of the stem on 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.
<i>getstems</i>	<p>Write the Contents of Members of Stems</p> <p>NetRexx</p>  <ul style="list-style-type: none"> • Input stream 0 should contain rexx objects containing stems. The getstems stage will output the contents of the stem on stream 0. If output stream 1 is connected, the root is placed there. Any severed streams will cause then stage to exit. Passing a non rexx stem object will cause the stage to exit with return code 13. • Pipes for NetRexx only.
<i>grep</i> <i>regex</i> 3.09	<p>Select Lines by a Regular Expresion</p>



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

hash
dict

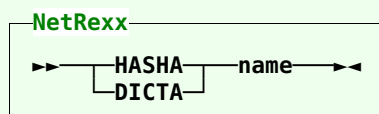
Read or Write a Dictionary



- Pipes for NetRexx only.

hasha
dicta

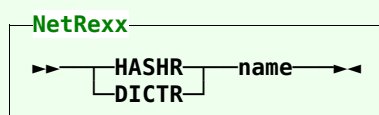
Write a Dictionary



- Pipes for NetRexx only.

hashr
dictr

Read a Dictionary

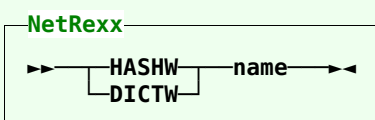


- Pipes for NetRexx only.

hashw

Write a Dictionary

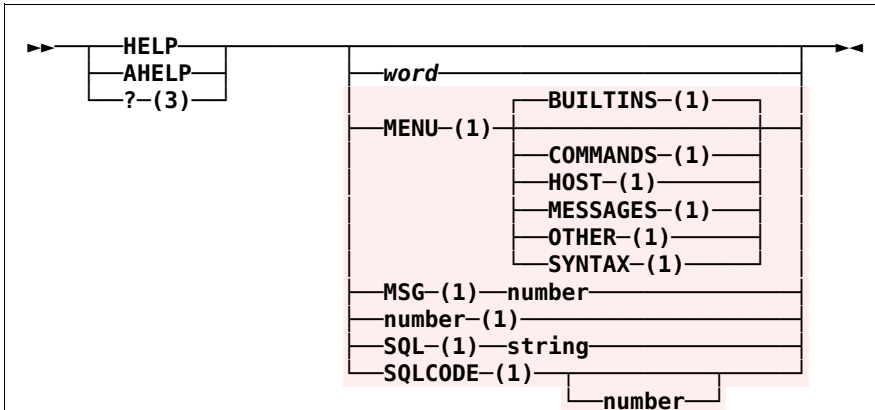
dictw



- Pipes for NetRexx only.

help
ahelp
?

Display Help for Pipelines



- (1) CMS Pipelines only. Not yet in NetRexx Pipelines.
- (2) If primary output is connected, lines are propagated, otherwise they are sent to the console by "say."
- (3) ? is the default pipeEnd character. Here it is useful only when a different pipeEnd is defined.

hfs
bfs

Read or Append File in the Hierarchical File System

- Not implemented in Netrexx Pipelines.

hfsdirectory
hfsdir
bfsdirectory
bfsdir

Read Contents of a Directory in a Hierarchical File System

- Not implemented in Netrexx Pipelines.

hfsquery
hfsq
bfsquery
bfsq

Write Information Obtained from OpenExtensions into the Pipeline

- Not implemented in Netrexx Pipelines.

hfsreplace
hfsrep
bfsreplace
bfsrep

Replace the Contents of a File in the Hierarchical File System

- Not implemented in Netrexx Pipelines.

hfsstate
hfsstat
bfsstate
bfsstat

Obtain Information about Files in the Hierarchical File System

- Not implemented in Netrexx Pipelines.

hfsxecute
hfsx
bfsxecute
bfsx

Issue OpenExtensions Requests

- Not implemented in Netrexx Pipelines.

hiasm

Interface to High Level Assembler

- Not implemented in Netrexx Pipelines.

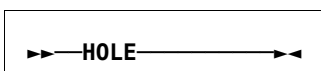
hiasmerr

Extract Assembler Error Messages from the SYSADATA File

- Not implemented in Netrexx Pipelines.

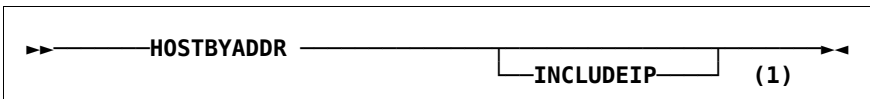
hole

Destroy Data



hostbyaddr
3.09

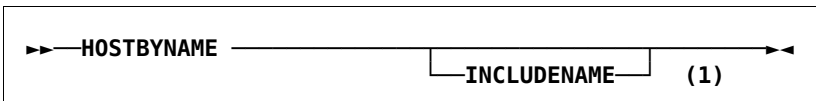
Resolve IP Address into Domain and Host Name



- (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

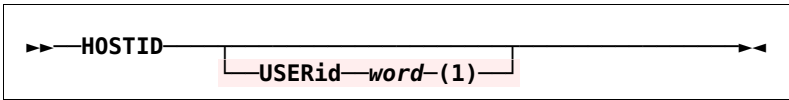
Resolve a Domain Name into an IP Address



- (1) Optional parameter not present in CMS Pipelines
- Arguments: INCLUDEDNAME - Also include the name of the host on output.
- Output: <hostname>/<ip address>
Example: `dns.google/8.8.8.8`

hostid
3.09

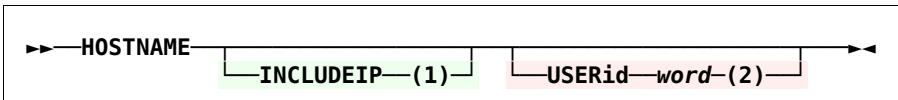
Write TCP/IP Default IP Address



- (1) The USERid option available under CMS Pipelines is not applicable and is ignored in NetRexx Pipelines

hostname
3.09

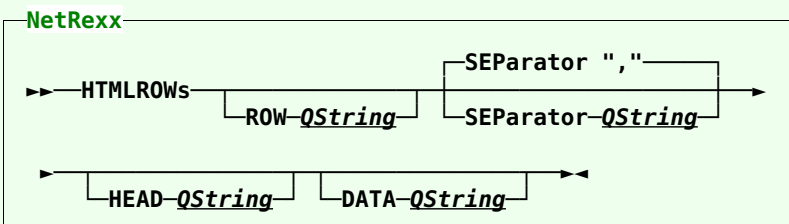
Write TCP/IP Host Name



- (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>

htmlrows
htmlrow
3.11

Convert rows to HTML format



- HTMLROWS reads rows from its primary input stream and writes them to its primary output stream, altering them to have the proper HTML tags for TABLE ROWS.
- I.e., it converts
abc,mnop,xyz
into
<tr><td>abc</td><td>mnop</td>
<td>xyz</td></tr>
- The SEPARATOR QString, by default the comma character, can be specified.
- There are options to put additional data inside the tags. This could be used for class or style tag options, for example.
 - ROW QString : puts its information into the <tr>-tags
 - DATA QString : puts its information into the <td>-tags
 - HEAD QString : puts its information into the <th>-tags (1)
- QString is a quoted string of characters. The quote marks may be either single or double, but must match. If there are no spaces in the string, the quote marks are optional.
- (1) If there is a HEAD option, the first row read has <th>-tags instead of <td>-tags. It must have a QString of at least "". Succeeding rows have the standard <td>-tags.

httpsplit

Split HTTP Data Stream

- Not implemented in Netrexx Pipelines.

iebcopy

Process IEBCOPY Data Format

- Not implemented in Netrexx Pipelines.

if

Process Records Conditionally

- Not implemented in Netrexx Pipelines.

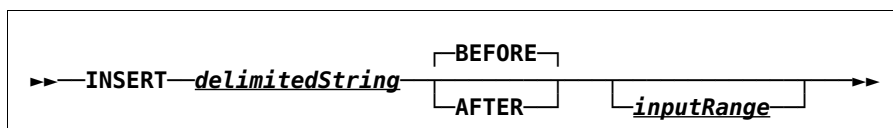
immcmd

Write the Argument String from Immediate Commands

- Not implemented in Netrexx Pipelines.

insert

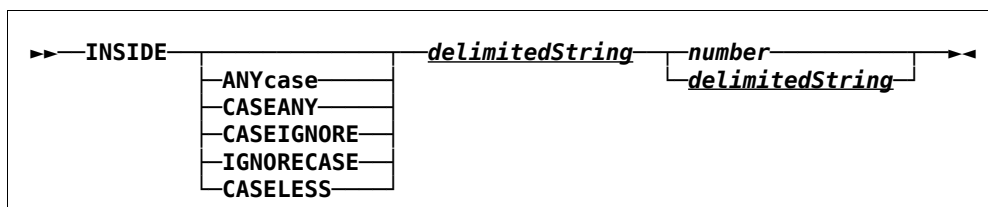
Insert String in Records



- 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



<i>instore</i>	Load the File into a storage Buffer <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>ip2socka</i>	Build sockaddr_in Structure <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>ispf</i>	Access ISPF Tables <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>jeremy</i>	Write Pipeline Status to the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>join</i>	Combine Records
<i>joincont</i>	Join Continuation Lines
<i>juxtapose</i>	Preface Record with Marker
<i>ldrtbls</i>	Resolve a Name from the CMS Loader Tables <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listcat</i>	Obtain Data Set Names <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listdsi</i>	Obtain Information about Data Sets <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
<i>listispf</i>	Read Directory of a Partitioned Data Set into the Pipeline <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.

<i>listpds</i>	Read Directory of a Partitioned Data Set into the Pipeline <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
----------------	---

<i>listzip</i>	List the Files in a Zipped File <p>NetRexx</p> <pre> --> LISTZIP --zipFileName--> </pre>
----------------	---

<i>literal</i>	Write the Argument String <pre> --> LITERAL --string--> </pre>
----------------	---

<i>locate</i> <i>locat</i> <i>loca</i> <i>loc</i> <i>lo</i> <i>l</i>	Select Lines that Contain a String <pre> --> Locate --ANYCase-- --CASEANY-- --CASEIGNORE-- --IGNORECASE-- --CASELESS-- --MIXED--(1) --ONEs--(1) --ZEROs--(1) --inputRanges-- --ANYof-- </pre> <pre> --delimitedString--> </pre> <p>(1) Not in NetRexx Pipelines, yet.</p> <ul style="list-style-type: none"> • [2] IBM documentation has this as "LOCATE" rather than "Locate". But the abbreviations work in both systems.
---	---

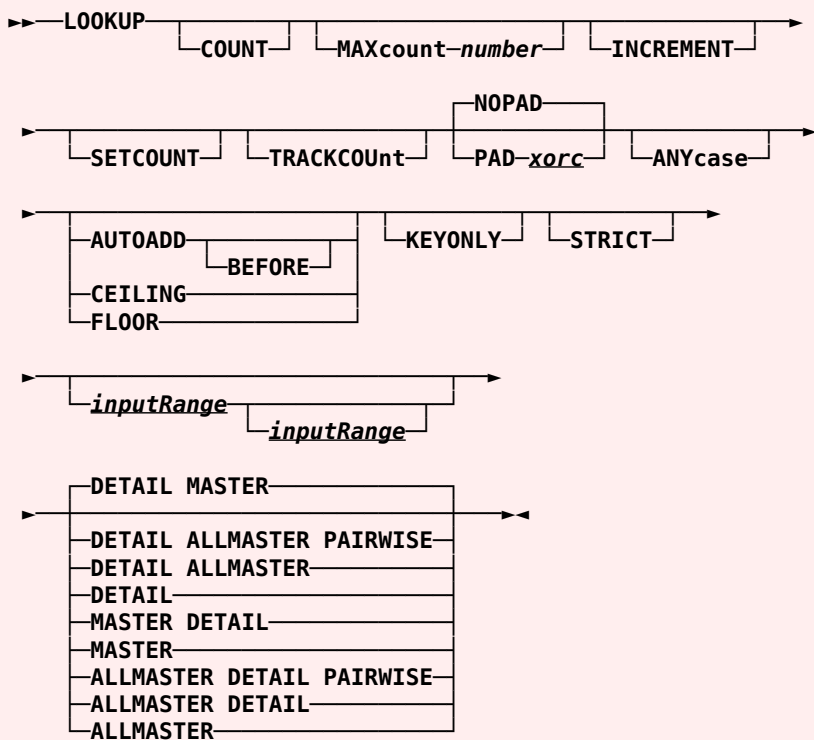
<i>lookup</i>	Find Records in a Reference Using a Key Field <p>NetRexx</p> <pre> --> LOOKUP --COUNT-- --ANYCASE-- --AUTOADD-- --BEFORE-- </pre> <pre> --KEYONLY-- --SETCOUNT-- --INCREMENT-- --TRACKCOUNT-- </pre> <pre> --inputRange-- --inputRange-- </pre> <pre> --DETAIL MASTER-- --DETAIL ALLMASTER PAIRWISE-- --DETAIL ALLMASTER-- --DETAIL-- --MASTER DETAIL-- --MASTER-- --ALLMASTER DETAIL PAIRWISE-- --ALLMASTER DETAIL-- --ALLMASTER-- </pre> <ul style="list-style-type: none"> • in stream 0 are detail records • in stream 1 are master records • in stream 2 adds to masters • in stream 3 delete from masters • • out stream 0 are matched records • out stream 1 are unmatched detail records • out stream 2 are unmatched or counted master records • out stream 3 deleted masters • out stream 4 duplicate masters • out stream 5 unmatched master deletes •
---------------	---

- lookup does not consider an unconnected output stream an error. It does propagate EOFs from output streams.

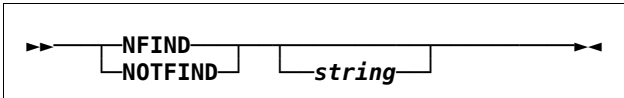
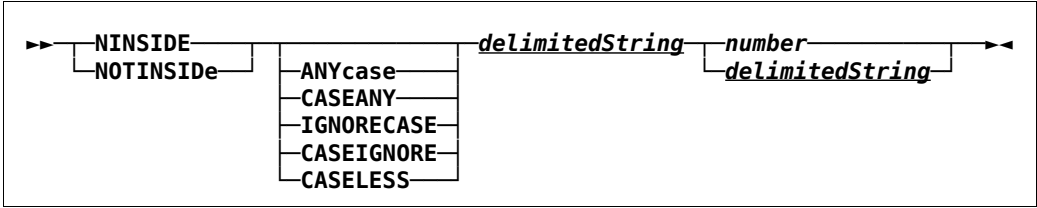
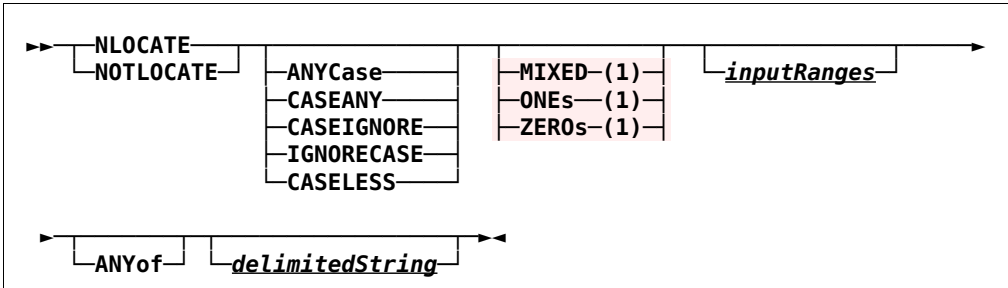
lookup

Find Records in a Reference Using a Key Field

CMS



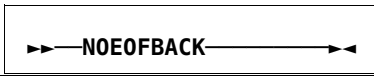
- in stream 0 are detail records
- in stream 1 are master records
- in stream 2 adds to masters
- in stream 3 delete from masters
-
- out stream 0 are matched records
- out stream 1 are unmatched detail records
- out stream 2 are unmatched or counted master records
- out stream 3 deleted masters
- out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

<i>maclib</i>	Generate a Macro Library from Stacked Members in a COPY File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mapmdisk</i>	Map Minidisks Into Data spaces <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mctoasa</i>	Convert CCW Operation Codes to ASA Carriage Control <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdiskblk</i>	Read or Write Minidisk Blocks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskrandom</i> <i>mdskrand</i>	Random Access a CMS File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskslow</i>	Read, Append to, or Create a CMS File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mdskupdate</i> <i>mdskupda</i>	Replace Records in a File on a Mode <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>members</i> <i>member</i>	Extract Members from a Partitioned Data Set <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>merge</i>	Merge Streams <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>mqsc</i>	Issue Commands to a WebSphere MQ Queue Manager <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>nfind</i> <i>notfind</i>	Select Lines by XEDIT NFind Logic 
<i>ninside</i> <i>notinsid</i> <i>notinside</i> 3.09	Select Records Not between Labels 
<i>nlocate</i> <i>notlocate</i>	Select Lines that Do Not Contain a String 

- (1) Not in NetRexx Pipelines, yet.

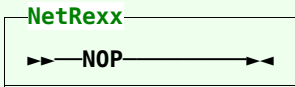
noEofBack

Pass Records and Ignore End-of-file on Output



nop

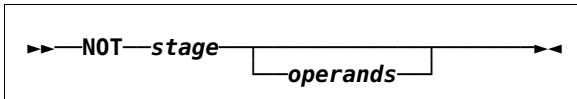
No Operation



- Pipes for NetRexx only.

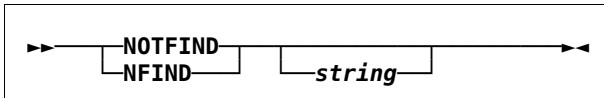
not

Run Stage with Output Streams Inverted



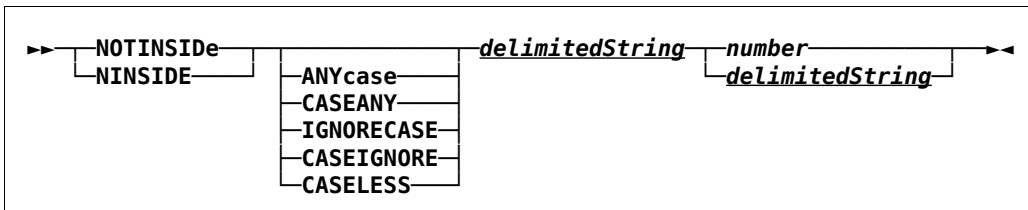
notfind
nfind

Select Lines by XEDIT NFind Logic



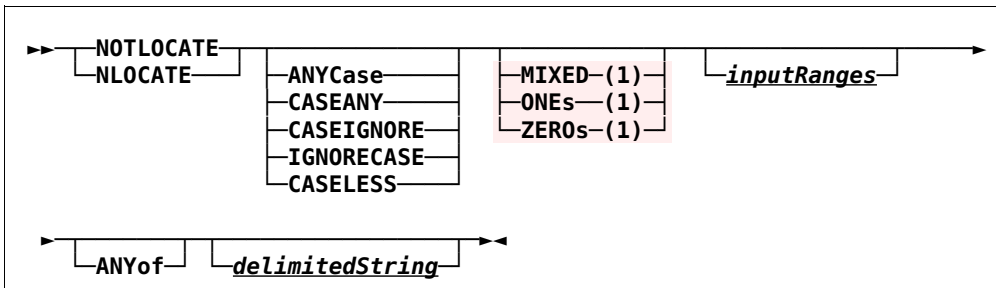
notininside
notininside
ninside

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 Netrex Pipelines.

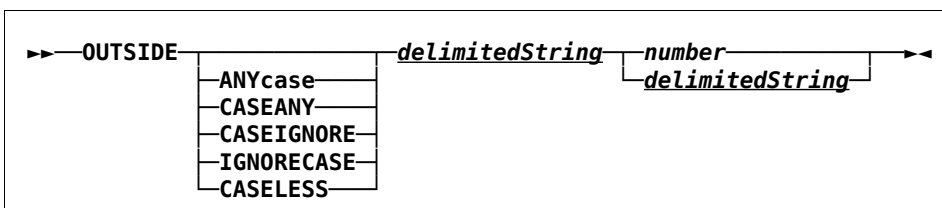
optcdj

Generate Table Reference Character (TRC)

- Not implemented in Netrex Pipelines.

outside

Select Records Not between Labels



outstore

Unload a File from a storage Buffer

- Not implemented in Netrex Pipelines.

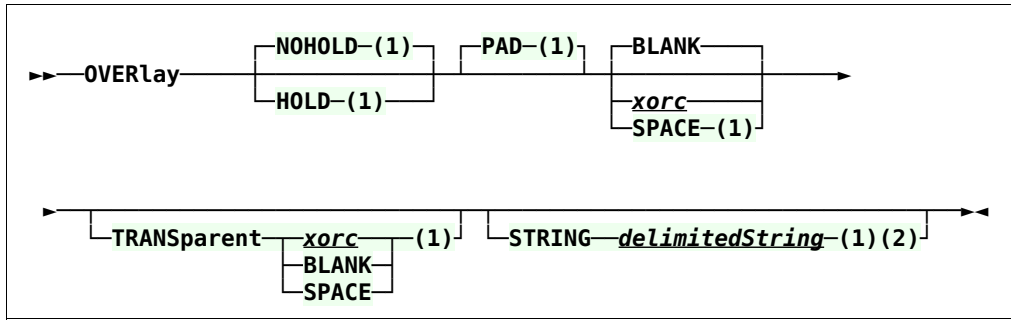
over

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

overstr

Process Overstruck Lines

- Not implemented in Netrex Pipelines.

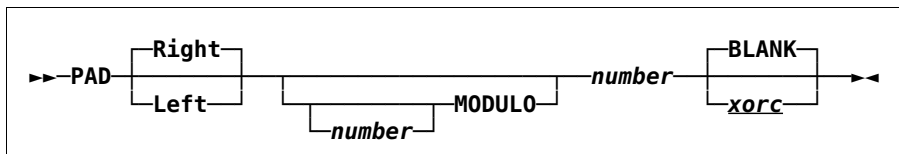
pack

Pack Records as Done by XEDIT and COPYFILE

- Not implemented in Netrex Pipelines.

pad

Expand Short Records



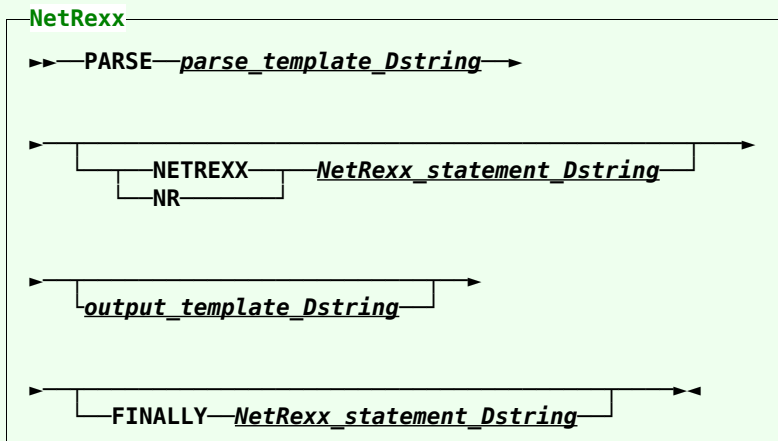
parcel

Parcel Input Stream Into Records

- Not implemented in Netrex Pipelines.

parse
4.06

Rearrange Contents of Records



- 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`.
- The two `NetRexx_statement_Dstrings` are single statements, or multiple statements separated by `;`'s.
 - The `_n` variables can be used and changed.
 - The string `\n` will split the string into separate output records.
 - The special indexed REXX variable `COUNTER[]` is also available in these `Dstrings`. This is specific to a `PARSE` stage, but persists between records. All the indexed values are initiated to 0. Both indexes and values can be strings.
 - This is powerful and has the possibility of doing damage to your pipe. You have been warned!
 - Due to the late compiling, at stage run time, debugging can be difficult. The reported line numbers have nothing to do with your code.
- If the `NR NetRexx_statement_Dstring` returns a value, it is used as the output instead of the optional `output_template_Dstring`.
- The `FINALLY's statement_Dstring` is executed after the last input record has been processed. The value returned is put out as an "extra" output record.
- (As of 4.05) Variable names of `"$n"` are deprecated, and can not be used with `NETREXX` or `FINALLY` options.
- `NetRexx Pipelines` only.
- .
- Examples:
 - `parse / 2 _1 +1/ /The second letter is "_1". __1 won't be changed./`
 - `parse /2 _1 +1/ NR /counter[1]=counter[1]+1; _9=counter[1]/ /_9/ FINALLY /return "Count:" _9/`
 - `PARSE /_1 2 _2 +1 _3/ , NR /if _2.datatype('L') then counter['c'] = counter['c'] + 1; _2 = _2.upper/ , /_1_2_3/ , FINALLY /return counter['c'] 'Changed to upper'/'`

pause

Signal a Pause Event

- Not implemented in `Netrexx Pipelines`.

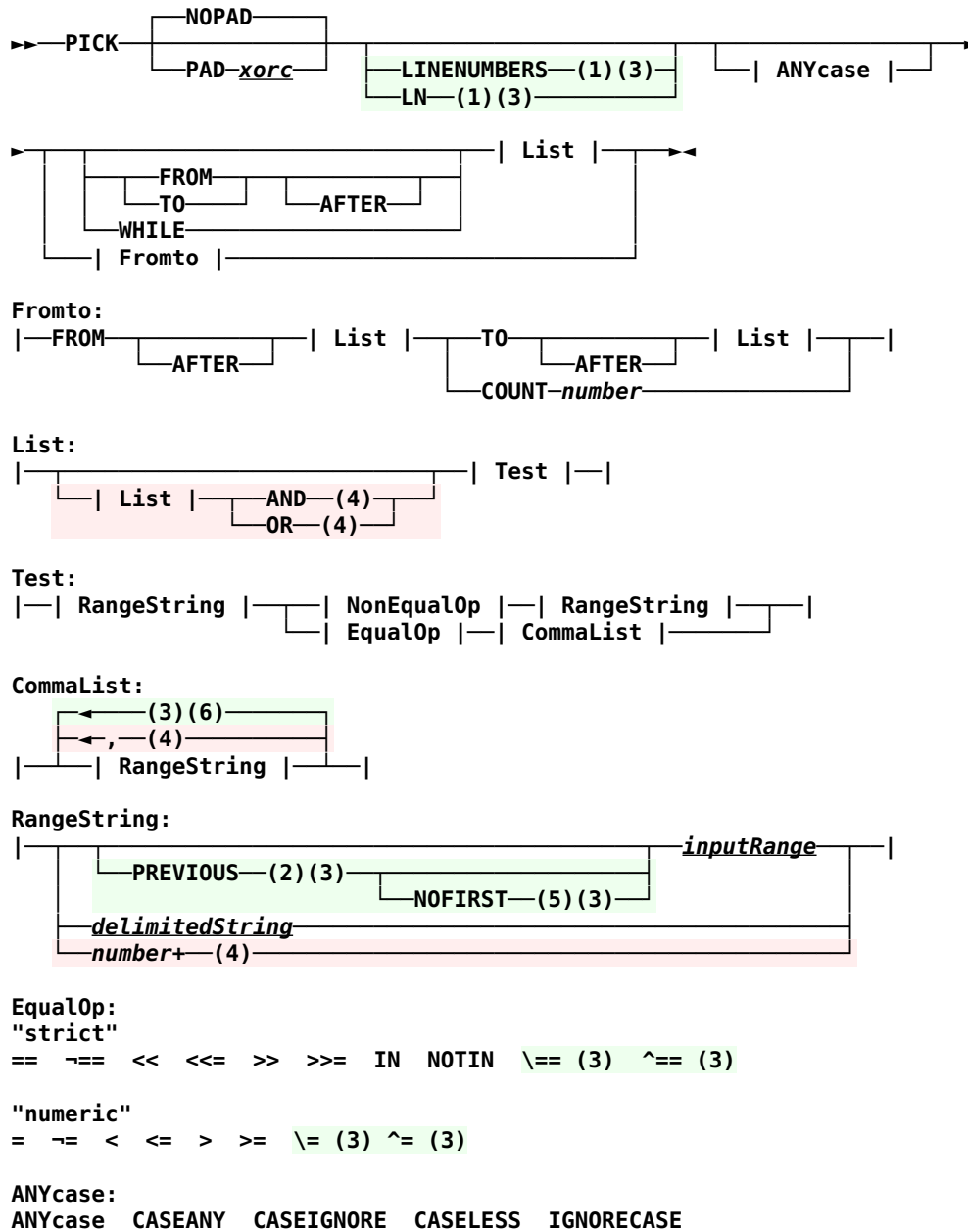
*pdsdirect
pds*

Write Directory Information from a CMS Simulated Partitioned Data Set

- Not implemented in `Netrexx Pipelines`.

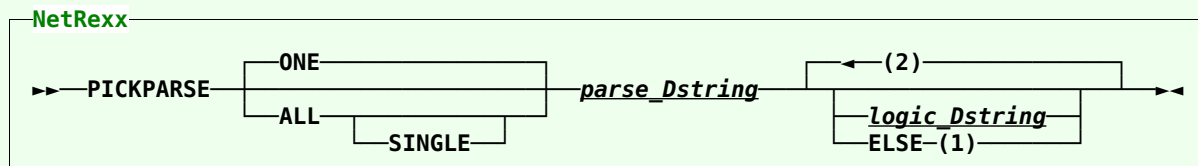
*pick
4.06*

Select Lines that Satisfy a Relation



- (1) NetRexx only. Inserts the original record number followed by a SPACE at the beginning of each output record.
- (2) NetRexx only. Uses the data from the previous record. Before the first record, this is Rexx "".
- (3) NetRexx Pipelines only. Not yet in CMS Pipelines.
- (4) CMS Pipelines only. Not yet in NetRexx Pipelines.
- (5) NetRexx Only. Uses first record data for first record instead of previous "".
- (6) CMS uses ",", NetRexx does not. CMS limits RangeStrings to right side, NetRexx allows them on the left, too. CMS also allows only == or \neq with RangeStrings. NetRexx permits any comparison op. NetRexx concatenates the several ranges for comparison.

Select Lines that Satisfy Relations using Rexx Parse



- 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 output streams (including an implied ELSE).
- Not implemented in CMS Pipelines.

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

Control Destructive Test of Records

- Not implemented in Netrexx Pipelines.

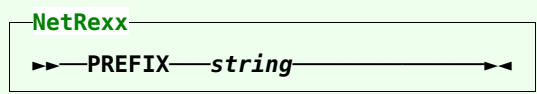
preface

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



- Blocks its primary input and executes stage supplied as an argument. The output from this stage are put to the primary output stream. When its complete the primary input is shorted.
- Not implemented in CMS Pipelines.

printmc

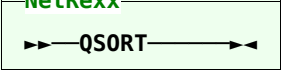
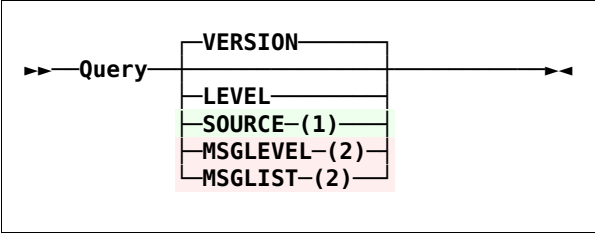
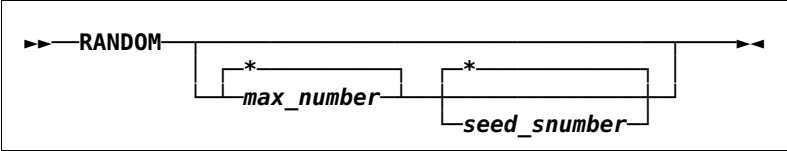
Print Lines

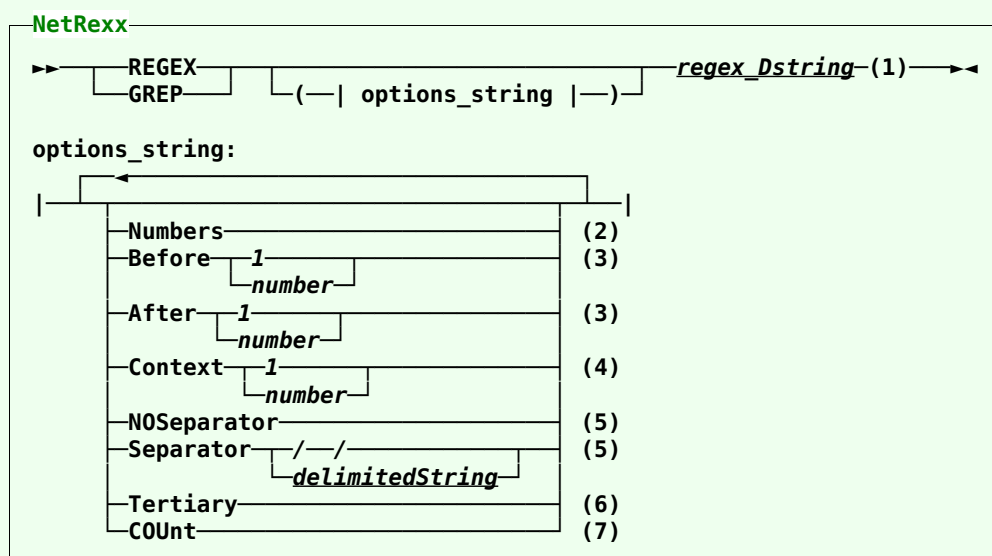
- Not implemented in Netrexx Pipelines.

punch

Punch Cards

- Not implemented in Netrexx Pipelines.

<i>qpdecode</i>	<p>Decode to Quoted-printable Format</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qpendcode</i>	<p>Encode to Quoted-printable Format</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qsam</i>	<p>Read or Write Physical Sequential Data Set through a DCB</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>qsort</i>	<p>Quick Order Records on Whole Length</p> <p>NetRexx</p>  <ul style="list-style-type: none"> • This sort routine is very basic. It uses sortRexx class, which implements the sortClass interface. To sort objects of classes other than Rexx requires that you implement another sortClass with a name beginning with 'sort'. • Not implemented in CMS Pipelines.
<i>query</i>	<p>Obtain Information From Pipelines</p>  <ul style="list-style-type: none"> • (1) Not CMS • (2) Not NetRexx Pipelines
<i>random</i> 3.09	<p>Generate Pseudorandom Numbers</p>  <ul style="list-style-type: none"> • NetRexx Pipelines will be a different sequence than CMS gives with the same seed.
<i>reader</i>	<p>Read from a Virtual Card Reader</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>readpds</i>	<p>Read Members from a Partitioned Data Set</p> <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>regex</i> <i>grep</i> 3.09	<p>Select Lines by a Regular Expression</p>



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

<i>retab</i>	Replace Runs of Blanks with Tabulate Characters <ul style="list-style-type: none"> • Not implemented in Netrex pipelines.
<i>reverse</i>	Reverse Contents of Records <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> ┌───┐ │ REVERSE ────────────────────> │ └───┘ </div>
<i>rexx</i>	Run a REXX Program to Process Data <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>rexxvars</i>	Retrieve Variables from a REXX or CLIST Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>runpipe</i>	Issue Pipelines, Intercepting Messages <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>scm</i>	Align REXX Comments <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>sec2greg</i>	Convert Seconds Since Epoch to Gregorian Timestamp

- Not implemented in Netrexx Pipelines.

select
4.07

Select Records using user logic

NetRexx



- Records are selected by evaluating the NetRexx T/F_Delimited_string, which consists of one or more NetRexx statements separated by ";", ending in a RETURN statement. The Dstring is placed in a method that supplies the record in the variable rec. The previous record is in prev. The method returns 1 to select the record to the primary output stream, or 0 to send it to the secondary stream.
- Alternatively with a Digit_NetRexx_Dstring, which evaluates to a 0 to 9 digit, the method can return the number of any output stream, but **NOTE: the primary and secondary numbers, 0 & 1 are reversed per the above logic.** Other streams have their corresponding number.
- Any other return value results in the record being discarded.
- Any record sent to a disconnected stream is discarded.
- This is powerful and has the possibility of doing damage to your pipe. You have been warned!
- Due to the late compiling, at stage run time, debugging can be difficult. The reported line numbers have nothing to do with your code.

Examples:

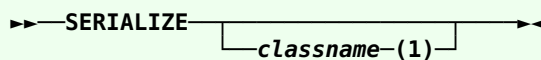
```
select /return rec.pos('2') > 0/
```

```
select /parse rec 2 r +1;parse prev 2 p +1; return r <> p/
```

serialize

Convert Objects to/from a Single Text String

NetRexx



- (1) classname if class is specified deserialize input to objects of this type - otherwise serialize input objects.
- (2) Pipes for NetRexx only.
- (3) For some reason readObject does not like more than one object network in its stream. Block multiple objects.
- (4) See examples/serialize_tests01.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.

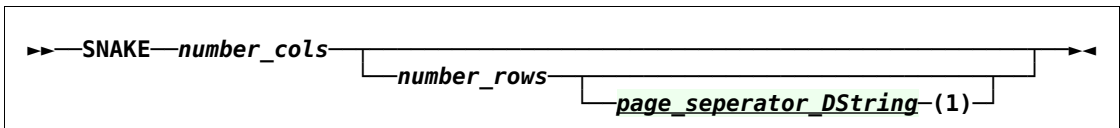
sfsupdate

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

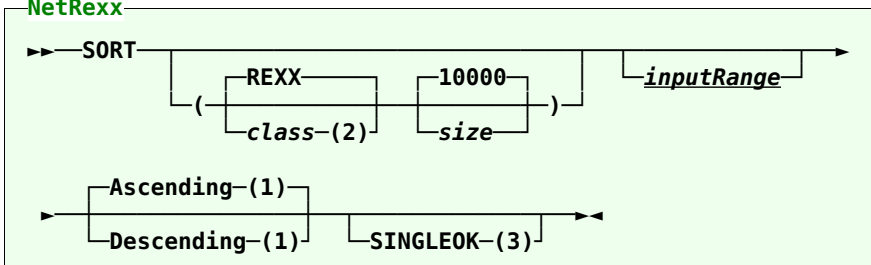
Format sockaddr_in Structure

- Not implemented in Netrexx Pipelines.

sort

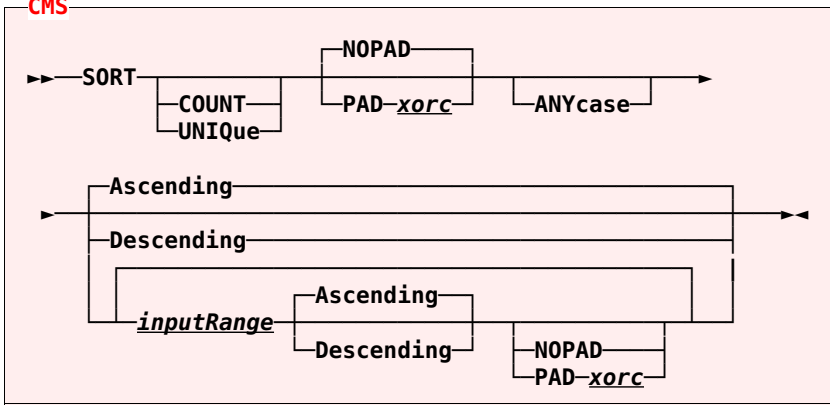
Order Records

NetRexx



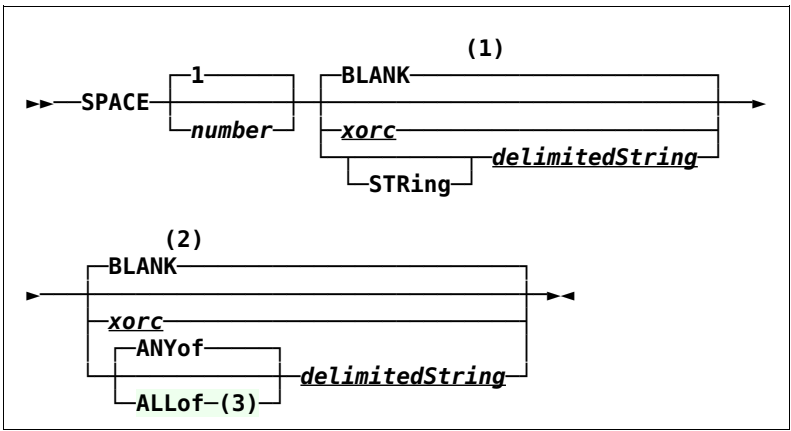
- (1) May come before inputRange, for backwards compatibility.
- (2) Requires that you implement another sortClass with a name beginning with 'sort'
- (3) Suppresses error message if only one record to sort for Rexx objects.
- Uses *sortClass* class as Interface Class for Generic Sort Objects and *sortRexx*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

CMS



space
3.09

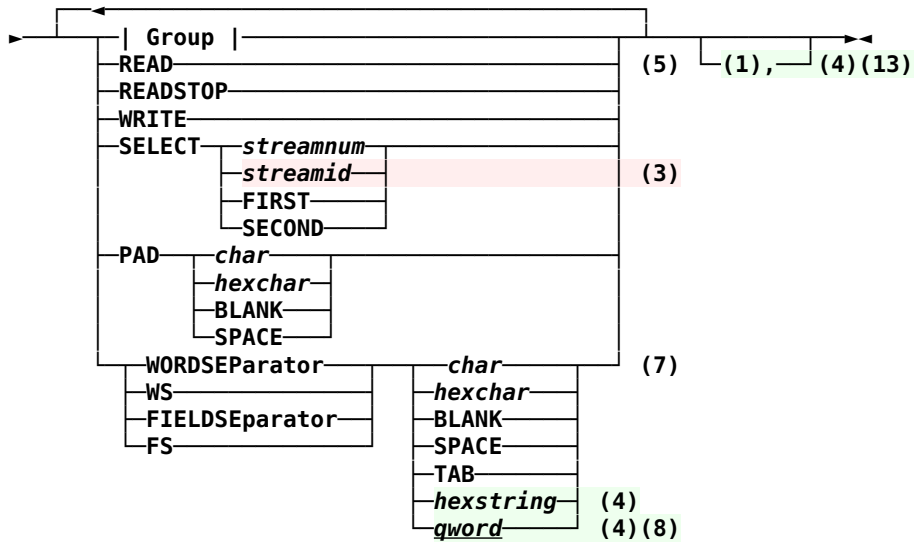
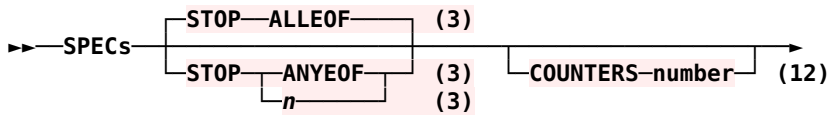
Space Words Like REXX



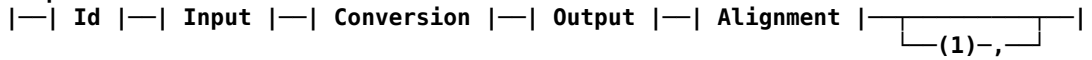
- (0) The order is the reverse of CHANGE!
- (1) the replacement char/string
- (2) the char/chars that will be stripped and replaced
- (3) NetRexx Pipelines only, not CMS. The dstring is treated as a single unit for stripping or replacing

spec
 specs
 4.06
 Fields and
 Separators
 NetRexx and
 COUNTERS
 comma
 separators

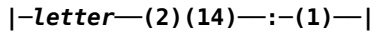
Rearrange Contents of Records



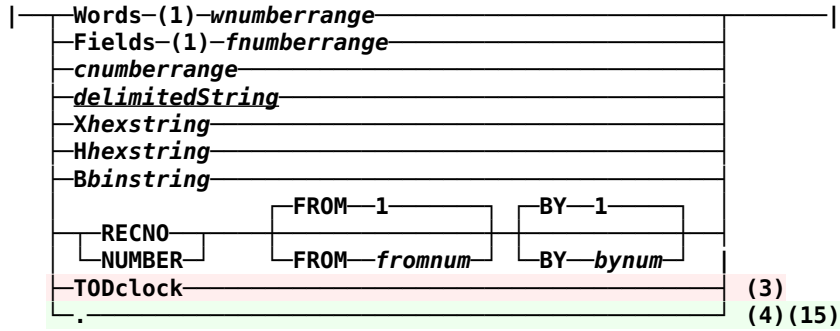
Group:



Id:



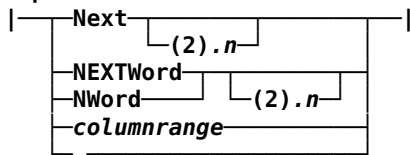
Input:



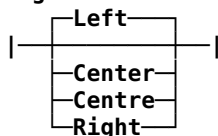
Conversion:

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		(3)
LOWER		(4)
UPPER		(4)
STRING		(4) (11)
NETREXX	delimitedString	(4) (10)
NR		(4)

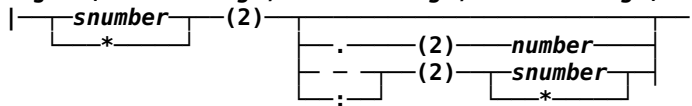
Output:



Alignment:



Ranges (cnumberrange, fnumberrange, wnumberrange):



- (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) NetRexx Pipelines only. 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.
- (7) CMS Pipelines, without documenting it, places this right by default NetRexx Pipelines follows the documentation and places this left by default. Specify the alignment you want to override these defaults.
- (8) A *qword* is an optionally quoted word, with single or double marks. If it contains

spaces or begins with a quote mark, it must be quoted. It can not start with a space (the quote mark will be considered a single character, and rest gibberish). If is unquoted and an even number of hexadecimal characters, it will be used as a hexchar or hexstring.

- (9) CMS has a mini-programming language built in. It uses Field Identifiers and Control Breaks, Counters, and Structured Data. NetRexx does not yet have any of these features.
- (10) The delimited string is any valid NetRexx code. **[Yes, you can get in trouble!]** It is put into a method and executed for each record. The selected input data is in the variable DATA. The returned string is output. The variable array COUNTER[] is available for your use. Unlike CMS, COUNTER is a full NetRexx variable object of type REXX. Each COUNTER is initially 0, but can hold any NetRexx value, including strings. COUNTERs are persistent for the life of the stage and are shared across all NETREXX converters in a stage. Index can be number or string. Fields identified by a fieldid are accessed as field["id"] when the quotes are required, the id is a single letter, case is respected.
- (11) The data is processed by the toString() method.
- (12) CMS Only. NetRexx ignores COUNTERS n; it has an unlimited number. See the NETREXX converter(10).
- (13) NetRexx Only. A comma may be used to separate groups for readability. It is ignored by the system.
- (14) Letter is a single character, a-zA-Z. Case is respected. BLANK is not permitted between letter and : . The resulting data is available in the NETREXX code as field["letter"].
- (15) NetRexx Only. A "." for input is "0.0", no data is selected. **data** is "".

Examples:

CMS Pipelines has built into the SPEC stage its own programming language. It is REXX-like-but-not-quite-REXX. For NetRexx Pipelines we have built in the worlds best scripting language: NetRexx. This gives all the power, but with a somewhat different syntax, of the CMS version. It is incorporated into the "Conversion" phase, with the key word NETREXX (or NR) and a delimitedString containing the NetRexx source. In running, this code is encapsulated in a method. The data selected in the Input phase is available as the variable **data**. And whatever is returned is passed to the Output phase of the stage. (As a convenience, if the last statement is not RETURN, the statement **"return data"** is automatically added.)

So all of these pipes work the same (giving

cba):

- pipe "literal abc | spec 1-* NR /return data.reverse/ 1 | cons"
- pipe "literal abc | spec 1-* NR @rev = data.reverse; return rev@ 1 | cons"
- pipe "literal abc | spec 1-* NR %data = data.reverse% 1 | cons"
- pipe "spec /abc/ NetRexx /data = data.reverse/ 1 | cons"

Using COUNTERS:

- pipe "literal abc | spec 1-* NR /counter[7]=counter[7]+1;ret=data.reverse counter[7];return ret/ 1 | cons" gives cba 1

Using Field Ids (and "." for output position).

In this case the variable data is not used at all.:

- pipe "literal abc | spec T:1-* . 1 NR @rev = field['T'].reverse; return rev@ 1 | cons"
- pipe "literal abc | spec T:1-* . t:1 NR @rev = field['T'].reverse; return rev@ 1 | cons" "T:" and "t:" are different.

Using comma to separate groups:

- pipe "literal abc | spec T:1-* 6-16, 1.2 NR /rev = field['T'].reverse; return rev/ 1.4 | cons" gives cba abc

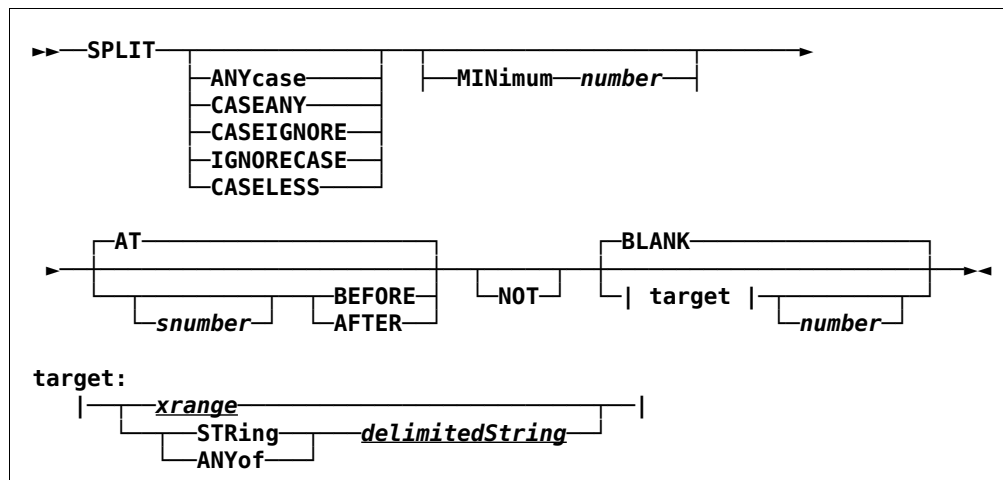
spill

Spill Long Lines at Word Boundaries

- Not implemented in Netrex Pipelines.

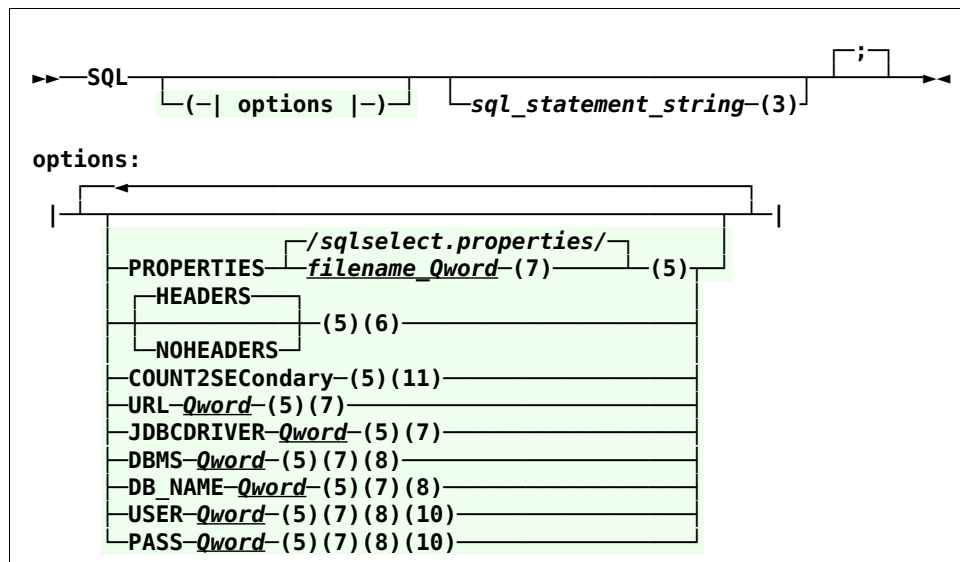
split

Split Records Relative to a Target



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.mssql.MssqlDriver
url=jdbc:mssql://localhost:1114/TESTDB
# the following are not needed for
some DBMS, ex: SQLite
user=db_user_name
pass=password_for_db
```

•

- if this file is not found default (compiled in) values are used
- (1) when using a sql select * (all columns) from the commandline, quote the query as in

```
java pipes.compiler (query) "sql
select * from dept | console"
```
- (2) the netrexx/jdbc combination is extremely case sensitive for column and table names
- (3) this sql_select_string executed, then statements are read from the primary input stream.
this is optional in NetRexx Pipelines only.
- (4) CMS does not use the stream input
- (5) NetRexx Pipelines only
- (6) CMS Pipelines is implied HEADERS only.
- (7) A Qword is an optionally quoted word. If it contains spaces, it must be quoted.
- (8) EXPERIMENTAL Subject to change. DBMS is the kind of database, e.g. SQLite. DB_name is the file name. 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, JDBC_DRIVER 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

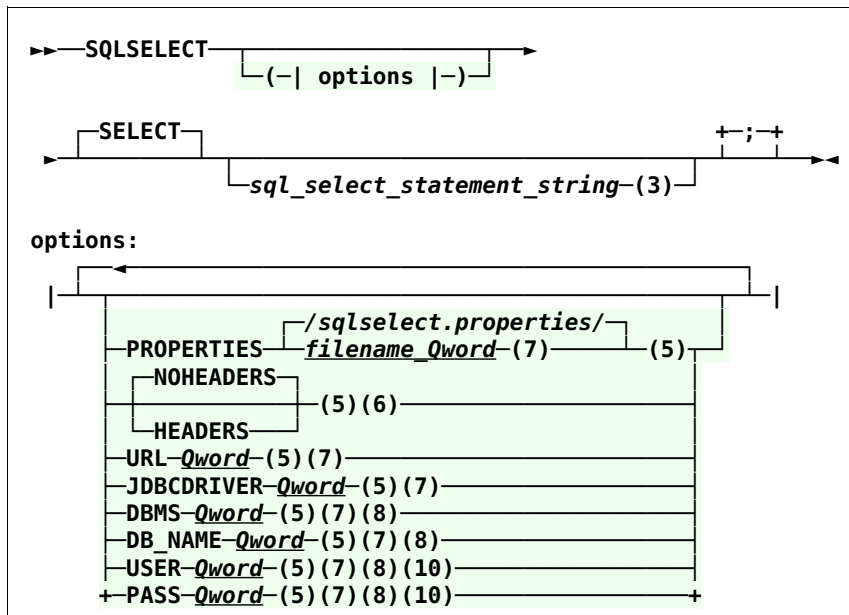
sqlcodes

Write the last 11 SQL Codes Received

- Not implemented in Netrexx Pipelines.

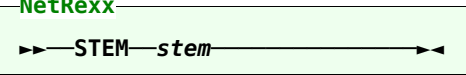
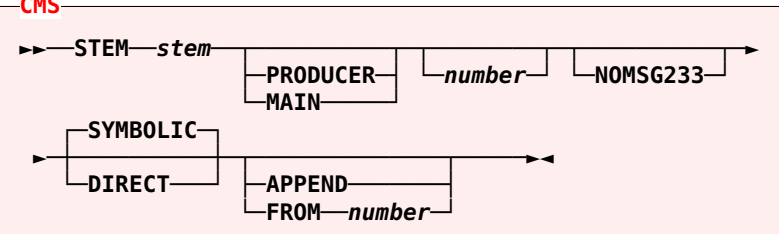
sqlselect

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 JDBC_DRIVER. 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, JDBC_DRIVER, 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

<i>stack</i>	Read or Write the Program Stack <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>starmon</i>	Write Records from the *MONITOR System Service <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>starmsg</i>	Write Lines from a CP System Service <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>starsys</i>	Write Lines from a Two-way CP System Service <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>state</i>	Provide Information about CMS Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>state</i>	Verify that Data Set Exists <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>stateg</i>	Provide Information about Writable CMS Files <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>stem</i>	Retrieve or Set Variables in a REXX or CLIST Variable Pool <div style="border: 1px solid black; padding: 5px; margin-top: 5px;"> <p>NetRexx</p>  <p>CMS</p>  </div>
<i>stfle</i>	Store Facilities List <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>storage</i>	Read or Write Virtual Machine Storage <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
<i>strasmfind</i>	Select Statements from an Assembler File as XEDIT Find

- Not implemented in Netrexx Pipelines.

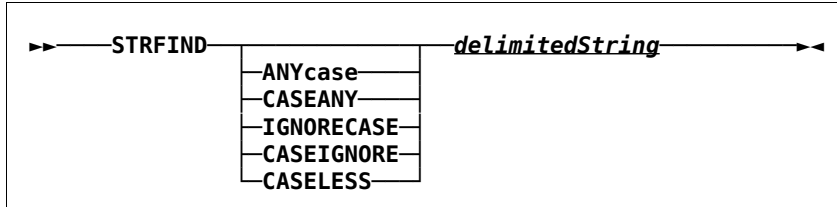
strasmnfind

Select Statements from an Assembler File as XEDIT NFind

- Not implemented in Netrexx Pipelines.

strfind

Select Lines by XEDIT Find Logic



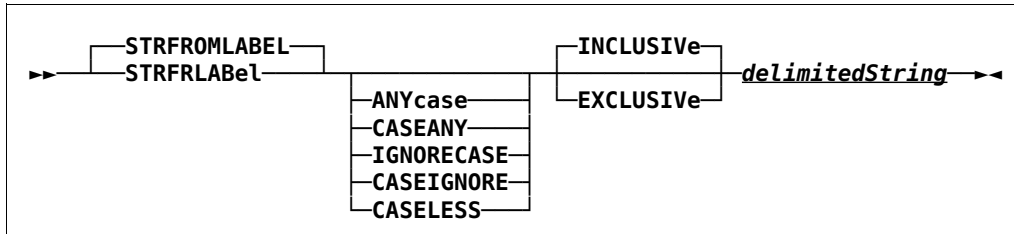
strfrlabel

strfrlabe

strfrlab

strfromlabel

Select Records from the First One with Leading String



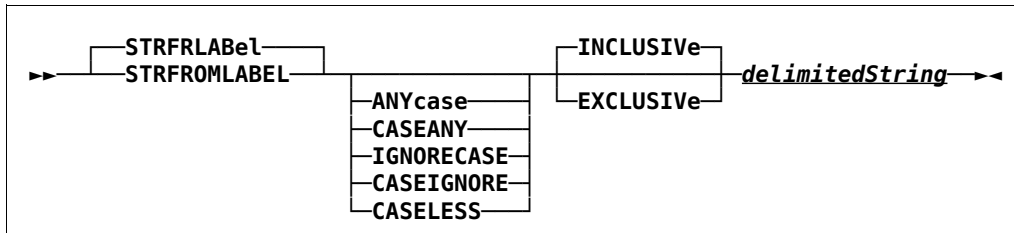
strfromlabel

strfrlabel

strfrlabe

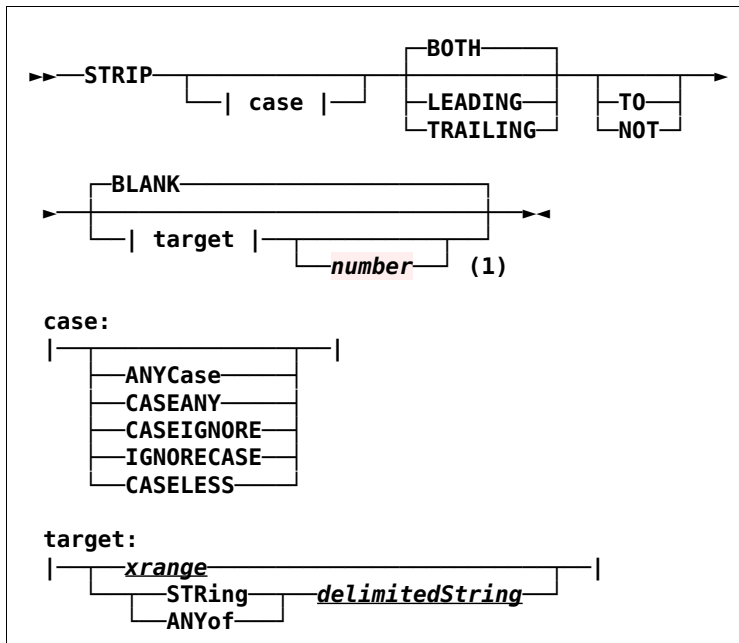
strfrlab

Select Records from the First One with Leading String



strip

Remove Leading or Trailing Characters



- (1) Not implemented in Netrexx Pipelines.

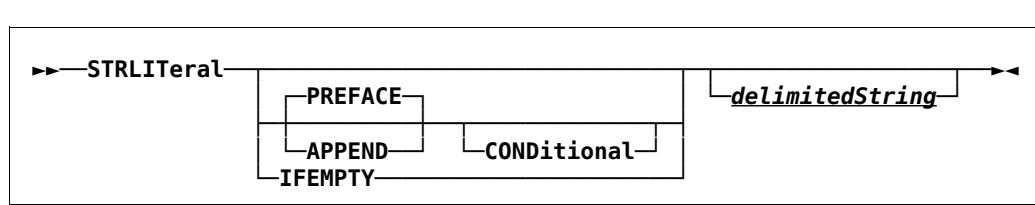
strliteral

strlitera

strliter

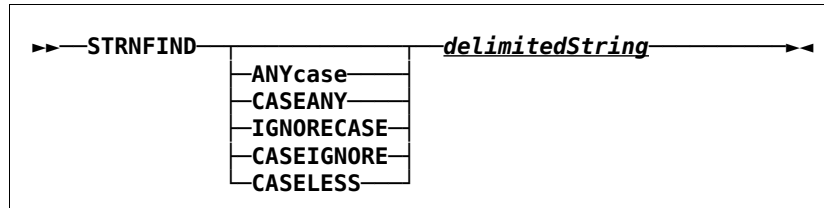
Write the Argument String

strlite
strlit



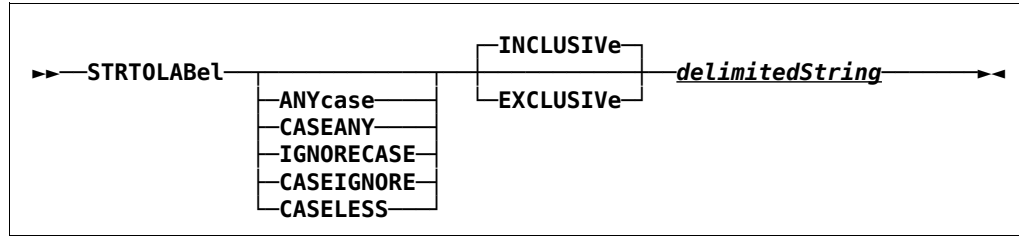
strnfind

Select Lines by XEDIT NFind Logic



strtolabel
strtolabe
strtolab

Select Records to the First One with Leading String



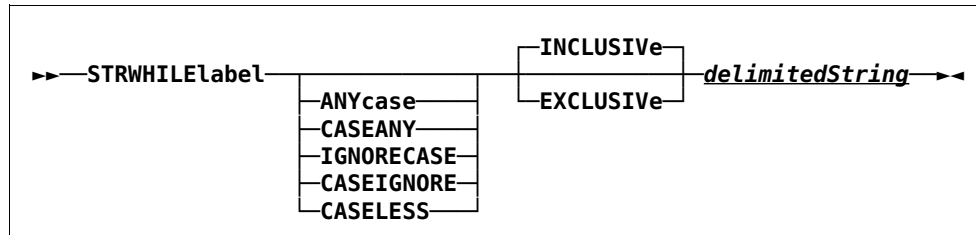
structure
struct

Manage Structure Definitions

- 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
substr

Write substring of record

- Not implemented in Netrexx Pipelines.

synchronise
sync
synchronize

Synchronise Records on Multiple Streams

- Not implemented in Netrexx Pipelines.

synchronize
sync
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.

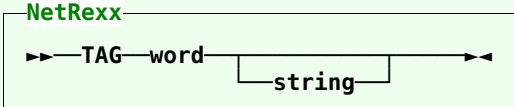
sysvar

Write System Variables to the Pipeline

- Not implemented in Netrexx Pipelines.

tag
3.11

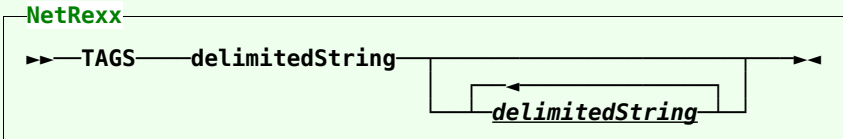
Surrounds Input Records with a HTML tag and its End Tag



- Outputs a record: <word string>, then passes through all records on its primary input, and finally a record: </word>.

tags
3.11

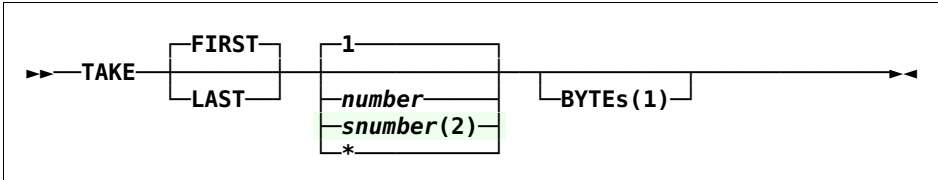
Surrounds Input Records with HTML tags and their End Tags



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

take

Select Records from the Beginning or End of the File



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

tape

Read or Write Tapes

- Not implemented in Netrexx Pipelines.

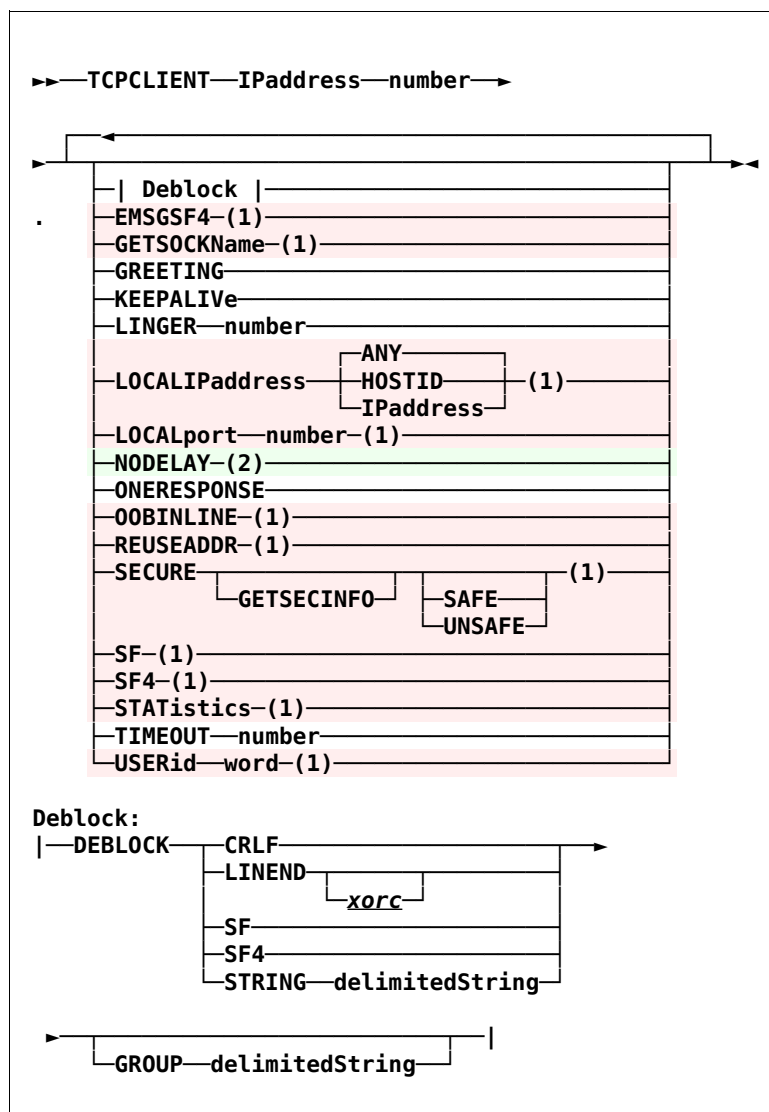
tcpchsum

Compute One's complement Checksum of a Message

- Not implemented in Netrexx Pipelines.

tcpclient

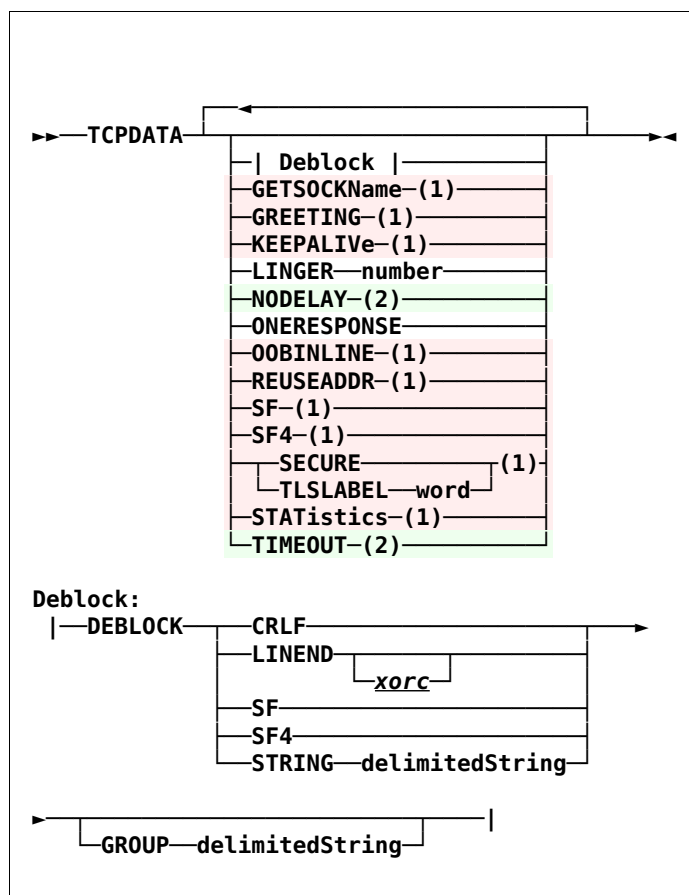
Connect to a TCP/IP Server and Exchange Data



- (1) CMS Pipelines Only.
- (2) NetRexx Pipelines Only.

The options implemented are similar to the CMS definition.

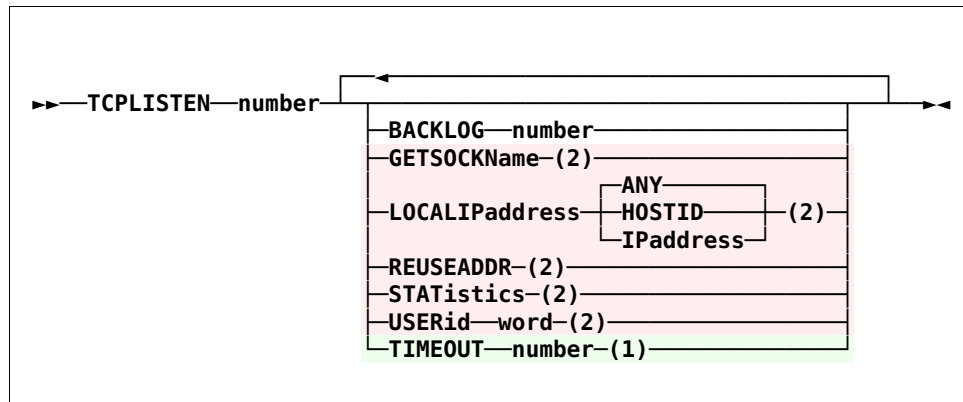
- `linger` - wait a bit before terminating the last read (units SECONDS)
- `timeout` - wait this long before timing reads out (units MS)
- `deblock` - If `deblock` is omitted a copy stage is used.
- `group` - similar to CMS. A delimited string containing a stage is expected. You can use a run of stages, but its is dangerous since you don't 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
- `onerespone` - synchronize cmds/replys



- Simple tcpdata implementation.
- (1) CMS Pipelines Only
- (2) NetRexx Pipelines Only
 - 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 it is dangerous since you do not know the stage separator character being used...
 - nodelay - use the nodelay option
 - onerresponse - synchronize requests/replies

tcplisten

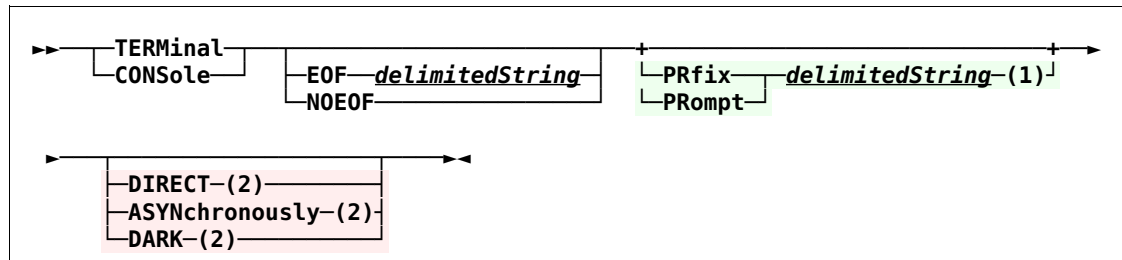
Listen on a TCP Port



- (1) NetRexx Pipelines only.
- (2) CMS Pipelines only.
- Simple tcplisten implementation. You can only supply the port and a timeout value. Its 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
 3.11

Read or Write the Terminal in Line Mode



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

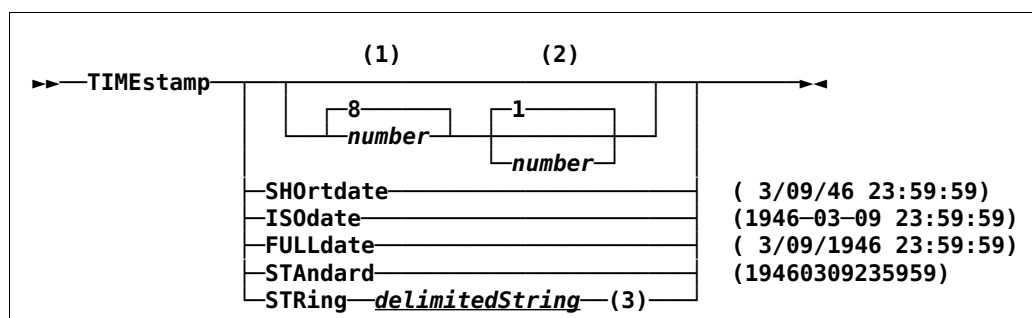
threeway

Split record three ways

- Not implemented in Netrexx Pipelines.

timestamp
 timestam
 timesta
 timest
 times
 time

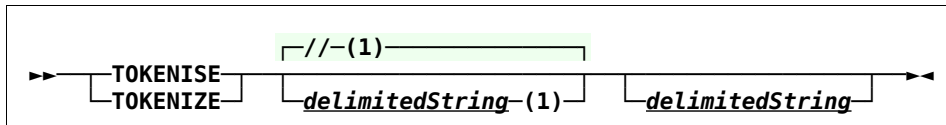
Prefix the Date and Time to Records



- (1) First character, from right, to include, <= 16
- (2) Count of characters to include. <= 16 - (1). Default = (1)
- (3)
 - %% A single %.
 - %Y Four digits year including century (0000-9999).
 - %y Two-digit year of century (00-99).
 - %m Two-digit month (01-12).
 - %n Two-digit month with initial zero changed to blank (1-12).
 - %d Two-digit day of month (01-31).
 - %e Two-digit day of month with initial zero changed to blank (1-31).
 - %j Julian day of year (001-366).
 - %H Hour, 24-hour clock (00-23).
 - %k Hour, 24-hour clock with leading zero blank (0-23).
 - %M Minute (00-59).
 - %S Second (00-60).
 - %F Equivalent to %Y-%m-%d (the ISO 8601 date format).
 - %T Short for %H:%M:%S.
 - %t Tens and hundredth of a second (00-99).

tokenise
tokenize

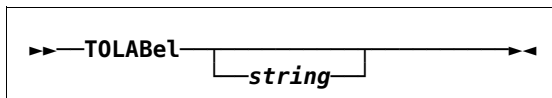
Tokenise Records



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

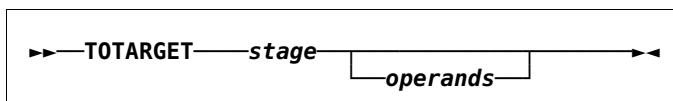
tolabel
tolabe
tolab

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**

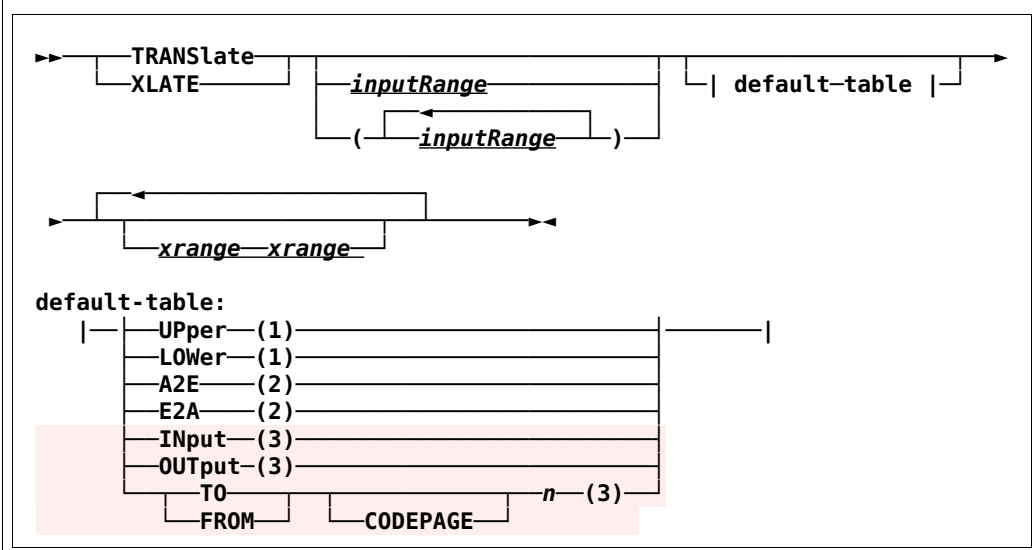
- Not implemented in Netrexx Pipelines.

trackxpan **Unsquish Tracks**

- Not implemented in Netrexx Pipelines.

translate
translat
transla
transl
trans
xlate

Transliterate Contents of Records



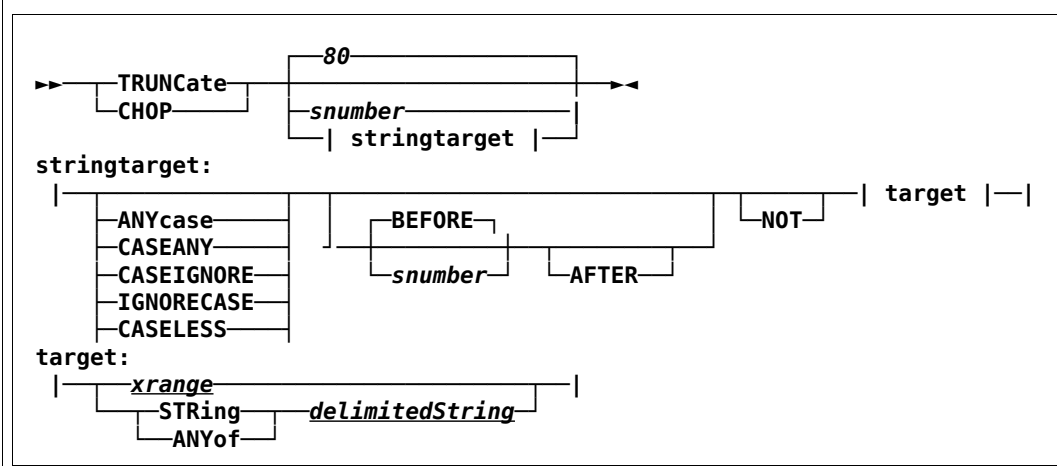
- Notes:
- (1) UTF-16 (ASCII) in NJPipes, probably EBCDIC in CMS.
 - (2) In Netrexx Pipelines. EBCDIC to ASCII or ASCII to EBCDIC. Maybe in CMS, the documentation is unclear.
 - (3) Not yet in NetRexx Pipelines
 - [4] NetRexx Pipelines only: The secondary input stream is not yet supported.

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.

<i>udp</i>	Read and Write an UDP Port <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
------------	---

<i>unique</i> <i>uniqu</i> <i>uniq</i>	Discard or Retain Duplicate Lines <p>uniqueRanges:</p> <pre> -----inputRange----- -----(-----inputRange-----)----- -----NOPAD----- -----PAD-xorc----- </pre>
--	---

<i>unpack</i>	Unpack a Packed File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
---------------	---

<i>untab</i>	Replace Tabulate Characters with Blanks <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
--------------	--

<i>unzip</i>	Extract Files From a ZIP Archive <p>Notes:</p> <ul style="list-style-type: none"> • (1) If filename is not specified, it is read from the primary input stream. Succeeding input objects are ignored. • The extracted file names are passed to the primary output stream. • Any existing files will be replaced. • This is NetRexx Pipelines only.
--------------	---

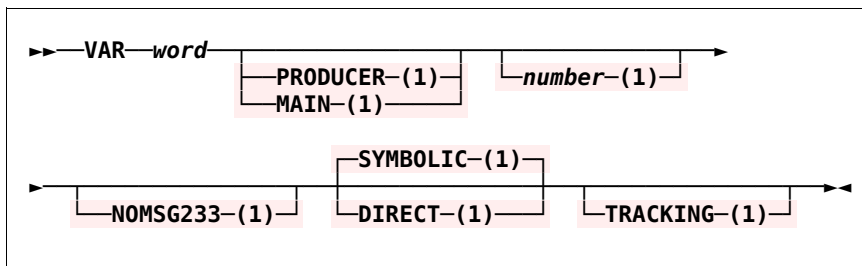
<i>update</i>	Apply an Update File <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
---------------	---

<i>urldeblock</i>	Process Universal Resource Locator <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
-------------------	---

<i>uro</i>	Write Unit Record Output <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
------------	---

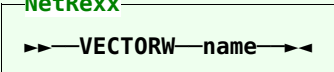
<i>utf</i>	Convert between UTF-8, UTF-16, and UTF-32 <ul style="list-style-type: none"> • Not implemented in Netrexx Pipelines.
------------	--

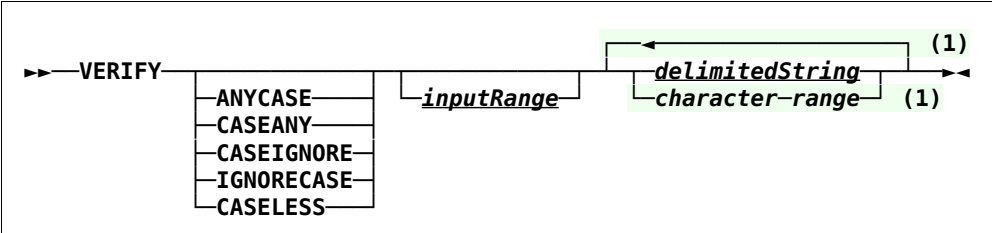
<i>var</i>	Retrieve or Set a Variable in a REXX or CLIST Variable Pool
------------	--



- In NetRexx Pipelines, this can only read vars. It must be the first stage in a pipe.
- (1) CMS Pipelines only.

<i>vardrop</i>	Drop Variables in a REXX Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>varfetch</i>	Fetch Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>varload</i>	Set Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>varover</i> 3.09	Write the Values of Stems <p>NetRexx</p> <ul style="list-style-type: none"> • NetRexx Pipelines only; not CMS Pipelines • If the secondary output stream is connected, the root is passed on it.
<i>varset</i>	Set Variables in a REXX or CLIST Variable Pool <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>vchar</i>	Recode Characters to Different Length <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
<i>vector</i>	Read or Write an Array of Vectors <p>NetRexx</p> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>vectora</i>	Add to an Array of Vectors <p>NetRexx</p> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>vectorr</i>	Read From an Array of Vectors <p>NetRexx</p> <ul style="list-style-type: none"> • Pipes for NetRexx only.
<i>vectorw</i>	Write to an Array of Vectors

	<p>NetRexx</p>  <ul style="list-style-type: none"> • Pipes for NetRexx only.
--	--

<p><i>verify</i> 3.09</p>	<p>Verify that Record Contains only Specified Characters</p>  <ul style="list-style-type: none"> • (1) NetRexx Pipelines only • (1) <i>character-range</i> is <i>xorc-xorc</i> • (1) Examples: A-Z 0-9 c-g a4-ba; 16-bit Unicode characters or hex numbers • (1) Any number greater than zero, any order, of delimitedStrings and character-ranges are allowed.
-------------------------------	--

<p><i>vmc</i></p>	<p>Write VMCF Reply</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
-------------------	---

<p><i>vmcdata</i></p>	<p>Receive, Reply, or Reject a Send or Send/receive Request</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
-----------------------	---

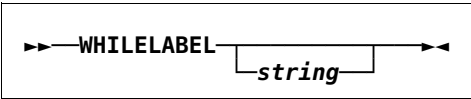
<p><i>vmclient</i></p>	<p>Send VMCF Requests</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
------------------------	---

<p><i>vmclisten</i></p>	<p>Listen for VMCF Requests</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
-------------------------	---

<p><i>waitdev</i></p>	<p>Wait for an Interrupt from a Device</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
-----------------------	--

<p><i>warp</i></p>	<p>Pipeline Wormhole</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
--------------------	--

<p><i>warplist</i></p>	<p>List Wormholes</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
------------------------	---

<p><i>whilelabel</i> 3.09</p>	<p>Select Run of Records with Leading String</p> 
-----------------------------------	---

<p><i>wildcard</i></p>	<p>Select Records Matching a Pattern</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
------------------------	--

<p><i>writepds</i></p>	<p>Store Members into a Partitioned Data Set</p> <ul style="list-style-type: none"> • Not implemented in Netrex Pipelines.
------------------------	--

<i>xab</i>	Read or Write External Attribute Buffers <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
------------	---

<i>xedit</i>	Read or Write a File in the XEDIT Ring <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
--------------	---

<i>xlate</i> <i>translate</i> <i>translat</i> <i>transla</i> <i>transl</i> <i>trans</i>	Transliterate Contents of Records <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p>default-table:</p> <ul style="list-style-type: none"> UPper (1) LOWer (1) A2E (2) E2A (2) INput (3) OUTput (3) TO FROM CODEPAGE n (3) </div> <p>Notes:</p> <ul style="list-style-type: none"> (1) UTF-16 (ASCII) in NJPipes, probably EBCDIC in CMS. (2) In Netrexx Pipelines. EBCDIC to ASCII or ASCII to EBCDIC. Maybe in CMS, the documentation is unclear. (3) Not yet in NetRexx Pipelines [4] NetRexx Pipelines only: The secondary input stream is not yet supported.
--	---

<i>xmsg</i>	Issue XEDIT Messages <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
-------------	---

<i>xpdhi</i>	Expand Highlighting to Space between Words <ul style="list-style-type: none"> Not implemented in Netrexx Pipelines.
--------------	---

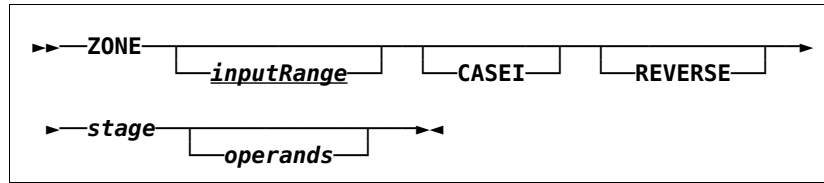
<i>xrange</i> 3.09	Write a Range of Characters <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> </div> <ul style="list-style-type: none"> NetRexx uses UTF-16 (ASCII) and CMS uses EBCDIC
-----------------------	---

<i>zip</i>	Add Files To a new ZIP Archive <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> </div> <ul style="list-style-type: none"> (1) name is the zip file name. If not provided, the first entry on the primary input stream is used. (1) If no extension is provided in name, ".ZIP" is added.
------------	--

- (1) Any existing file is replaced.
- Subsequent records on the primary input stream are filenames to be added to the Zip archive file.
- File names added passed out on primary out stream.
- NetRexx Pipelines only.

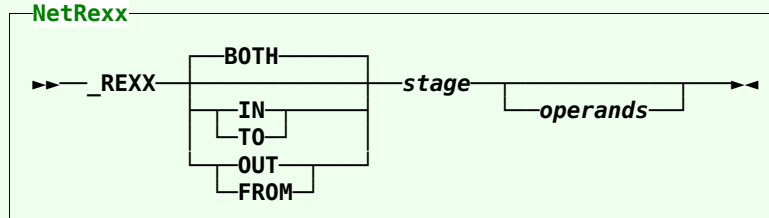
zone

Run Selection Stage on Subset of Input Record



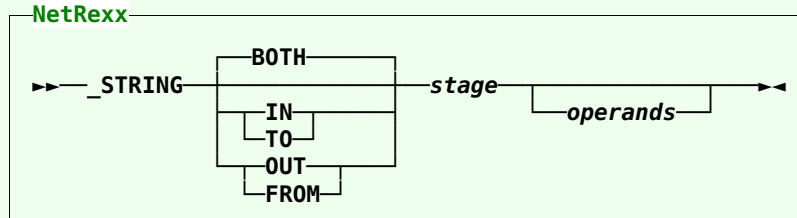
_rex

Cast Input and/or Output of a Stage to Type REXX



_string

Cast Input and/or Output of a Stage to Type String



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 similar effect.

List of Tables

1	Device drivers	13
2	Record selection	14
3	Filters	15
4	Other stages	15

Index

NetRexx Workspace, 8
Rexx, 17, 21, 23
This, 22
arg, 9, 22, 23
catch, 11, 18, 22-24
class, 9-11, 17, 18, 21-25
do, 10, 11, 17, 23, 24
end, 11, 17-24
exit, 11, 17, 18, 22-24
extends, 10, 11, 17, 22-24
final, 10, 11, 22-24
for, 8
forever, 10, 11, 17, 23, 24
import, 10, 11, 17, 22-24
loop, 10, 11, 17, 21-24
method, 10, 11, 17, 18, 21-24
over, 22
public, 22-24
rex, 10, 11
say, 21
signal, 17
signals, 18, 23
static, 22, 23
to, 21
where, 26

built-in stages, 31

deadlock, 20
device driver, 4

filter stages, 14

label, 19

multi-stream pipelines, 18

nrws, 8
nrws.input, 8

other stages, 15

pipe command, 9
pipes runner, 27

record selection stages, 14

stall, 20

