# EnhancedPIPE

<sichler@rumms.uni-mannheim.de>

**COLLABORATORS**

| | | | |
|---|---|---|---|
| | *TITLE* :<br><br>EnhancedPIPE | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | <sichler@rumms.uni-mannheim.de> | January 2, 2023 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# EnhancedPIPE

## 1.1 Enhanced PIPE - Users Guide

Enhanced PIPE

-------------

by Stefan Sichler

*** FREEWARE ***

Introduction What is it?

Installation How to put it on my computer?

Usage How to use it?

## 1.2 introduction

The EnhancedPIPE EPIPE: is an AmigaDOS device, that works quite similar to the original PIPE: device.

But EPIPE: has some special features that help to better control the behaviour of the pipe.

Though some features that can only be used withing an application, most of them can be used from a normal CLI.

These special features are activated by not only specifiing a pipe name when opening an EPIPE: filehandle, but also specifiing a so-called qualifier extension .

## 1.3 installation

If you are using AmigaOS 1.x, follow the installation instructions for AmigaOS 2.x, but I am not sure if it works under 1.3.

If you are using AmigaOS 2.x, execute the following commands:

copy EPIPE-Handler to L:

join DEVS:Mountlist EPIPE-Mountlist to DEVS:Mountlist

and reboot.

If you are using AmigaOS 3.x, execute the following commands:

copy EPIPE-Handler to L:

copy EPIPE EPIPE.info to DEVS:DOSDrivers/

and reboot.

If you like to develop software using EPIPE:, you should install the file EPIPE.h anywhere in your INCLUDE: directory. I myself have it in INCLUDE:User/EPIPE.h.

## 1.4 usage

An EPIPE: pipe can not only have a read and a write filehandle, but some other filehandles too. You tell EPIPE: what filehandle you want to open by specifiing qualifier filename extension .

Nevertheless, there can only be one read and one write filehandle.

EPIPE: does no internal data buffering, so a write request of a write filehandle can only be completed, if there are appropriate read requests.

Read requests, in turn, can only be completed if there are write requests, or the pipe signals End Of File (EOF).

An EOF: signal is generated, when the standard write filehandle of a pipe gets closed.

Btw., a standard read filehandle can only be opened if a write filehandle is already opened, unless you use the "wait" qualifier extension .

Examples: ----------------------------

Open a shell and enter:

echo >EPIPE:Testpipe "Hallo!"

Outputs "Hallo!" to the pipe "Testpipe" by using a standard write filehandle.

If you open a second shell and enter:

type EPIPE:Testpipe

You will get:

Hallo!

Then on both shells, the prompt will return.

---------------------------------------

Open a shell and enter:

type EPIPE:Testpipe .wait

This will open a read filehandle.

Then, open a second shell and enter:

echo >EPIPE:Testpipe .nobreak "This is the first line..."

echo >EPIPE:Testpipe "...and this the second one!"

After this the "type" command on the first shell will return and print:

This is the first line...

...and this the second one!

## 1.5 qualifiers

Qualifier filename extensions

<pipename> (read/write)

----------------------

Opens a standard read or write filehandle.

Read():

Reads data from the pipe.

Write():

Writes data to the pipe.

<pipename>.wait (read)

----------------------

Opens a read filehandle, that will not return if there is currently no write filehandle opened in the pipe.

Read():

Reads data from the pipe.

<pipename>.nobreak (write)

--------------------------

Opens a write filehandle, that will send no EOF signal to the pipe if it gets closed again.

Write():

Writes data to the pipe.

<pipename>.force (write)

------------------------

Opens a force filehandle.

Write():

Whenever you do a Write(), a pending read request in the pipe will be returned.

<pipename>.trigger (read)

-------------------------

Opens a trigger filehandle.

Read():

Your read request is returned whenever there is a pending read request in the pipe.

<pipename>.signal (write)

-------------------------

Opens a trigger filehandle.

Write():

Whenever you write at least one byte, EPIPE assumes that the first byte of your write data contains a signal number obtained by your program by AllocSignal(). So when a read request shows up in the pipe, EPIPE sends a signal to your task via Signal() using the given signal number.

<pipename>.info (read)

----------------------

Opens an info filehandle.

Read():

You should read in EPIPE_INFO_LENGTH (defined in EPIPE.h) bytes.

Then will get a newline and null terminated string that will contain...

a 'R' if a readfilehandle is currently opened in the pipe, followed by the total length of the request and the number of bytes already read in decimal format a ’W’ if a writefilehandle is currently opened, followed by the total length

and the number of bytes already written in decimal format

a ’T’ if a triggerfilehandle is currenty opened.

a ’F’ if a forcefilehandle is currenty opened.

Example:

"R377,10 T\n"

This means that the pipe has currently an read request that wants to read 377 bytes,

10 bytes are already transferred to it, and there is an open triggerfilehandle.

ATTENTION: This filehandle can ONLY BE READ ONCE, then you have to reopen it.

<pipename>.bininfo (read)

-------------------------

Opens an info filehandle.

Read():

You should read in EPIPE_BININFO_LENGTH (defined in EPIPE.h) bytes.

You get a EPIPE_bininfo structure, that contains information about the current

state of the pipe.

See EPIPE.h.

## 1.6 Who am I?

You can contact me at:

sichler@rumms.uni-mannheim.de

or stefan@ds.domino.de