NAME

zoem – macro processor for the Zoem macro/programming language.

SYNOPSIS

zoem [-i <file name>[.azm] (*entry file name*)**] [-I** <file name> (*entry file name*)**] [-o** <file name> (*output file name*)**] [-d** <device> (*set device key*)**]**

zoem
(enter interactive mode - happens when none of **-i**, **-I**, **-o** is given)

zoem -i <file name>[.azm] (*entry file name*) **-I** <file name> (*entry file name*) **[-o** <file name> (*output file name*)**] [-d** <device> (*set device key*)**] [-x** (*enter interactive mode on error*)**] [-s** <key>[=<val>] (*set key to val*)**] [-e** <any> (*evaluate any, exit*)**] [-E** <any> (*evaluate any, proceed*)**] [-chunk-size** <num> (*process chunks of size num*)**] [--trace** (*trace mode, default*)**] [--trace-all-long** (*long trace mode*)**] [--trace-all-short** (*short trace mode*)**] [--trace-regex** (*trace regexes*)**] [-trace** k (*trace mode, explicit*)**] [--stats** (*show symbol table stats after run*)**] [--split** (*assume \writeto usage, set __split__*)**] [--stress-write** (*make \write#3 recover*)**] [--unsafe** (*prompt for \system#3*)**] [--unsafe-silent** (*simply allow \system#3*)**] [-allow** cmd1[:cmdx]+ (*allowable commands*)**] [--system-honor** (*require \system#3 to succeed*)**] [-nuser** k (*user dict stack size*)**] [-nenv** k (*environment dict stack size*)**] [-nsegment** k (*maximum simple nesting depth*)**] [-nstack** k (*maximum eval nesting depth*)**] [-buser** (*initial user dict capacity*)**] [-bzoem** (*initial zoem dict capacity*)**] [-tl** k (*tab length*)**] [-l** <str> (*list items*)**] [-h** (*show options*)**] [--apropos** (*show options*)**]**

DESCRIPTION

Zoem is a macro/programming language. It is fully described in the *Zoem User Manual* (zum.html), currently available in HTML only. This manual page documents the zoem processor, not the zoem language.

If the input file is specified using the **-i** option and is a regular file (i.e. not STDIN - which is specified by using a single hyphen), it must have the extension `.azm`. This extension can but need not be specified. The zoem key **__fnbase__** will be set to the file base name stripped of the `.azm` extension and any leading path components. If the input file is specified using the **-I** option, no extension is assumed, and **__fnbase__** is set to the file base name, period. The file base name is the file name with any leading path components stripped away.

If neither **-i** nor **-o** is specified, zoem enters interactive mode. Zoem should fully recover from any error it encounters in the input. If you find an exception to this rule, consider filing a bug report. In interactive mode, zoem start interpreting once it encounters a line containing a single dot. Zoem's input behaviour can be modified by setting the key **__parmode__**. See the section

SESSION MACROS for the details. In interactive mode, zoem does *not* preprocess the interactive input, implying that it does not accept inline files and it does not recognize comments. Both types of sequence will generate syntax errors. Finally, readline editing and history retrieval can be used in interactive mode provided that they are available on the system. This means that the input lines can be retrieved, edited, and discarded with a wide range of cursor positioning and text manipulation commands.

From within the entry file and included files it is possible to open and write to arbitrary files using the **\write#3** primitive. Arbitrary files can be read in various modes using the **\dofile#2** macro (providing four different modes with respect to file existence and output), **\finsert#1**, and **\zinsert#1**. Zoem will write the default output to a single file, the name of which is either specified by the **-o** option, or constructed as described below. Zoem can split the default output among multiple files. This is governed from within the input files by issuing **\writeto#1** calls. Refer to the **--split** option and the *Zoem User Manual*.

If none of **-i** or **-o** is given, then zoem will enter interactive mode. In this mode, zoem interprets by default chunks of text that are ended by a single dot on a line of its own. This can be useful for testing or debugging. In interactive mode, zoem should recover from any failure it encounters. Interactive mode can also be accessed from within a file by issuing `\zinsert{stdia}`, and it can be triggered as the mode to enter should an error occur (by adding the **-x** option to the command line).

If **-o** is given and **-i** is not, zoem reads input from STDIN.

If **-i** is given and **-o** is not, zoem will construct an output file name as follows. If the **-d** option was used with argument `<dev>`, zoem will write to the file which results from expanding **__fnbase__.<dev>**. Otherwise, zoem writes to (the expansion of) **__fnbase__.ozm**.

For **-i** and **-o**, the argument − is interpreted as respectively `stdin` and `stdout`.

OPTIONS

-i <file name>[.azm] (*entry file name*)
Specify the entry file name. The file must have the `.azm` extension, but it need not be specified.

-I <file name>[.azm] (*entry file name*)
Specify the entry file name, without restrictions on the file name.

-o <file name> (*output file name*)
Specify the output file name.

-d <device> (*set key __device__*)
Set the key **__device__** to `<device>`.

-x (*enter interactive mode on error*)
The afterlife option. If zoem encounters an error during regular processing, it will emit error messages as usual, and then enter interactive mode. This allows you e.g. to inspect the values of keys used or defined within the problematic area.

-s <key>[=<val>] (*set key to val*)
Set the key **\key** to **val** if present, **1** otherwise. Any type of key can be set, including keys taking arguments and keys surrounded in quotes. Beware of the shell's quote and backslash interpolation. Currently **val** is not evaluated, so appending or prepending to a key is not possible.

-e <any> (*evaluate any, exit*)
This causes zoem to evaluate `<any>`, write any result text to stdout, and exit.

-E <any> (*evaluate any, proceed*)
This causes zoem to evaluate `<any>`, write any result text to stdout, and proceed e.g. with the entry file or an interactive session.

-chunk-size <num> (*process chunks of size num*)
Zoem reads its input in chunks. It fully processes a chunk before moving on with the next one. This option defines the (minimum) size of the chunks. The size or count of the chunks does not at all affect zoem's output. The default minimum chunk size equals one megabyte.

Zoem will read files in their entirety before further processing if **-chunk-size 0** is specified.

Zoem does not chunk input files arbitrarily. It will append to a chunk until it is in the outermost scope (not contained within any block) and the chunk will end with a line that was fully read.

Consequently, if e.g. a file contains a block (delimited by balanced curlies) spanning the entire file then zoem is forced to read it in its entirety.

--trace (*trace mode, default*)
Trace in default mode.

--trace-all-long (*long trace mode*)
Sets on *most* trace options in long mode. Trace options `xxx` not set have their own **--trace-xxx** entry (see below).

--trace-all-short (*short trace mode*)
Sets on *most* trace options in short mode. Trace options `xxx` not set have their own **--trace-xxx** entry (see below).

--trace-keys (*trace keys*)
Trace keys.

--trace-regex (*trace regexes*)
Trace regexes (i.e. the **\inspect#4** primitive).

-trace k (*trace mode, explicit*)
Set trace options by adding their representing bits.

--stress-write (*stress test using write*)
This makes **\write#3** recover from errors. It is a special purpose option used for creating zoem stress test suites, such as `stress.azm` in the zoem distribution `/examples` subdirectory.

--unsafe (*prompt for \system#3*)
--unsafe-silent (*simply allow \system#3*)
-allow cmd1[:cmdx]+ (*allowable commands*)
With **--unsafe** system calls are allowed but the user is prompted for each invocation. The command and its arguments (if any) are shown, but the STDIN information (if any) is withheld. With **--unsafe-silent** system calls are allowed and the user is never prompted.

Use **-allow** *str* or **--allow**=*str* to specify a list of allowable commands, as a string in which the commands are separated by colons.

--system-honor (*require \system#3 to succeed*)
With this option any `\system#3` failure (for whatever reason, including safe behaviour) is regarded as a zoem failure. By default, failing system calls are ignored under either safe mode, unsafe mode (--unsafe), or silent unsafe mode (--unsafe-silent).

--split (*assume split output*)
This assumes zoem input that allows output to multiple files (e.g. chapters). It sets the default output stream to `stdout` (anticipating custom output redirection with **\writeto#1**) and sets the session macro **__split__** to `1`.

--stats (*show symbol table stats after run*)
Show symbol table chacteristics. Symbol tables are maintained as hashes.

-tl k (*set tab length*)
Set the tab length. HTML output can be indented according to nesting structure, using tabs which are expanded to simple spaces. By default, the tab length is zero, meaning that no indent is shown. The maximum value the tab length can be set to is four.

-nsegment k (*level of macro nesting allowed*)
-nstack k (*stack count*)
-nuser k (*user dictionary stack size*)
-nenv k (*environment dictionary stack size*)
-buser k (*initial user dict capacity*)
-bzoem k (*initial zoem dict capacity*)
Probably needed only if you have some obscure and extreme use for zoem. The segment limit applies to simple nesting of macros. The stack limit applies to nesting of macros that evaluate an argument before use. Each such evaluation creates a new stack. The user limit applies to **\push{user}**, the env limit applies to the nesting level of environments (started with `\begin#2`). The user dict capacity pertains to the initial number of buckets allocated for user and environment dictionaries, and the zoem dict capacity pertains to the dictionary containing the zoem primitives.

-l <str> (*list items*)
List items identified by `<str>`. It can be any of **all**, **filter**. **legend**, **builtin**, **session**, **trace**, or **zoem**, Multiple identifiers can be joined in a string, e.g. `-l legendzoem` prints a legend followed by a listing of zoem primitives.

-h (*show options*)
Show short synopsis of options.

SESSION MACROS

__parmode__
This macro affects zoem's read behaviour in interactive mode. It can be set on the command line using the **-s** option. Bits that can be set:

```
1    chomp newlines (remove the newline character)
2    skip empty newlines
4    read paragraphs (an empty line triggers input read)
8    newlines can be escaped using a backslash
16   read large paragraphs (a single dot on a line
     triggers input read)
```

__device__

The current output device, set by the command line option **-d**. The **man** and **faq** packages support **html** and **roff** as its values.

__fnbase__
The base name of the input file name. Leading path components are stripped away. If the **-i** option is used the input file is required to have the `.azm` suffix. In that case the suffix is also stripped to obtain the base name.

__fnentry__
The name of the entry file.

__fnin__
The file currently being processed.

__fnout__
The name of the default output file.

__fnpath__
The leading component of the input file name, possibly empty.

__fnup__
The file that included the current file, if applicable.

__fnwrite__
This key is set by `\write#3` to its first argument. It can be used by macros that are expanded during evaluation of the third argument. Possible usage is to branch on the name of the *write* output stream. For example a file called *index.html* may be treated differently from other files. The key is deleted afterwards. Nested invocations of `\write#3` may corrupt the status of this key.

__ia__
The input/output separator used in interactive mode.

__line__
The line number in the file currently being processed. This number will be correct for any invocation outside the scope of a macro. For any invocation within a macro the result will be the line number of the closing curly of the outermost containing macro. The following

```
\__line__
\__line__
\__line__
\group{
\__line__
\group{\__line__}
\__line__}
```

Results in

```
1
2
3
7
7
7
```

__searchpath__
A vararg containing a list of paths to search when a file is to be included/imported/read/loaded. When you start zoem, this key should contain the location of the **man.zmm** and **faq.zmm** package files. It is advisable not to overwrite this key but to append to it instead.

__zoemstat__
Set to one of `ok`, `towel` (that one is a bit lame), or `error` by either the interpreter, an occurrence of `\catch#2`, or `\try#1`.

__zoemput__
Set by `\try#1` to the possibly truncated result of processing its argument.

__lc__
Expands to a left curly. It is hard to find a need for this − the zoem stress suite uses it to generate a particular syntax error at a deeper interpretation level.

__rc__
Expands to a right curly.

THE SET MODIFIERS

The `\set#3` primitive allows a `{modes}{<modifiers>}` directive in its first argument. Here *<modifiers>* can be a combination of single-letter modifiers, each described below.

a append to the key, do not overwrite, create if not existing.
c conditionally; only set if not already defined.
e existing; update existing key, possibly in lower dictionary.
g global; set in the global (bottom) user dictionary.
u unary; do not interpret vararg in `<any>` as key-value list (data keys only)
v vararg; interpret vararg in `<any>` as key-value list (regular keys only).
w warn if key exists (like `\def#2` and `\defx#2`).
x expand argument (like `\setx#2` and `\defx#2`).

THE INSPECT SUBLANGUAGE

The `\inspect#4` primitive takes four arguments. The languages accepted by the first two arguments are described below. The third argument is a replacement string or a replacement macro accepting back-references (supplied as an anonymous macro). The fourth argument is the data to be processed.

arg 1
Is a vararg. Currently it accepts a single key *mods* for which the value should be a comma-separated list over the words *posix*, *icase*, *dotall*, *iter-lines iter-args*, *match-once*, *discard-nmp*, *discard-nil-out*, *discard-miss*, *count-matches*. Alternatively repeated use of *mods* is allowed.

arg 2
Is a regular expression. Tilde patterns are expanded according to all of the ZOEM, UNIX, and REGEX schemes. Refer to **TILDE EXPANSION** for these.

The third argument is a constant string or an anonymous key, the fourth argument is data.

THE TR SUBLANGUAGE

The `\tr#2` primitive takes two arguments. The first argument contains key-value pairs. The accepted keys are *from* and *to* which must always occur together, and *delete* and *squash*. The values of these keys must be valid *translation* specifications. This primitive transforms the data in the second argument by successively applying translation, deletion and squashing in that order. Only the transformations that are needed need be specified.

Translation specifications are subjected to UNIX tilde expansion as described below.

The syntax accepted by translation specifications is almost fully compliant with the syntax accepted by **tr**(1), with three exceptions. First, repeats are introduced as `[*a*20]` rather than `[a*20]`. Second, ranges can (for now) only be entered as `X-Y`, not as `[X-Y]`. `X` and `Y` *can* be entered in either octal or hexadecimal notation (see further below). As an additional feature, the magic repeat operator `[*a#]` stops on both class and range boundaries. Character specifications can be complemented by preceding them with the caret `^`.

Specifications may contain ranges of characters such as `a-z` and `0-9`. Posix character classes are allowed. The available classes are

```
[:alnum:]
[:alpha:]
[:cntrl:]
[:digit:]
[:graph:]
[:lower:]
[:print:]
[:punct:]
[:space:]
[:upper:]
[:xdigit:]
```

Characters can be specified using octal notation, e.g. `\012` encodes the newline. Use `\173` for the opening curly, `\175` for the closing curly, `\134` for the backslash, and `\036` for the caret if it is the first character in a specification. *DON'T* use `\\`, `\{`, or `\}` in this case! Hexadecimal notation is written as `\x7b` (the left curly in this instance).

See **EXAMPLES** for an example of `tr#2` usage.

TILDE EXPANSION

Some primitives interface with UNIX libraries that require backslash escape sequences to encode certain tokens or characters. The backslash is special in zoem too and without further measures it can become very cumbersome to encode the correct escape sequences as it is not always clear which tokens should be escaped or unprotected at what point. It is especially difficult to handle the zoem characters with special meaning, `{`, `}` and `\`.

The two primitives under consideration are **\inspect#4** and **\tr#2**. Both treat the tilde as an additional escape character for certain arguments (as documented in the user manual). These arguments are subjected to tilde expansion, where the tilde and the character it proceeds are translated to a new character or character sequence. There are three different sets of tilde escapes, ZOEM, UNIX and REGEX escapes. **\tr#2** only accepts UNIX escapes, **\inspect#4** accepts all. Tilde expansion is always the last processing step before strings are passed on to external libraries.

The ZOEM scheme contains some convenience escapes, such as `~E` to encode a double backslash.

ZOEM tilde expansion

meta sequence	replacement
~~	~
~E	\\
~e	\
~I	\{
~J	\}
~x	\x
~i	{
~j	}

The zoem tr specification language accepts `\x**` as hexadecimal notation, e.g. `\x0a` denotes a newline in the ASCII character set.

UNIX tilde expansion

```
 meta sequence   replacement
.----------------------------.
|     ~a      |     \a      |
|     ~b      |     \b      |
|     ~f      |     \f      |
|     ~n      |     \n      |
|     ~r      |     \r      |
|     ~t      |     \t      |
|     ~v      |     \v      |
|     ~0      |     \0      |
|     ~1      |     \1      |
|     ~2      |     \2      |
|     ~3      |     \3      |
`----------------------------'
```

REGEX tilde expansion

```
 meta sequence   replacement
.----------------------------.
|     ~^      |     \^      |
|     ~.      |     \.      |
|     ~[      |     \[      |
|     ~$      |     \$      |
|     ~(      |     \(      |
|     ~)      |     \)      |
|     ~|      |     \|      |
|     ~*      |     \*      |
|     ~+      |     \+      |
|     ~?      |     \?      |
`----------------------------'
```

ENVIRONMENT

The environment variable ZOEMSEARCHPATH may contain a colon and/or whitespace separated list of paths. It will be used when searching for files included via one of the `dofile` aliases `\input`, `\import`, `\read`, and `\load`. Note that the zoem macro `\__searchpath__` contains the location where the zoem macro files were copied at the time of installation of zoem.

DIAGNOSTICS

On error, Zoem prints a file name and a line number to which it was able to trace the error. The number reported is the same as the one stored in the session macro `\__line__`. For an error-trigering macro which is not nested within another macro the line number should be correct. For a macro that does occur nested within another macro the line number will be the line number of the closing curly in the outermost containing macro.

If in despair, use one of the tracing modes, **--trace-keys** is one of the first to come to mind. Another possibility is to supply the **-x** option.

BUGS

No known bugs. **\inspect#4** has not received thorough stress-testing, and the more esoteric parts of its interface will probably change.

SEE ALSO

Aephea is a document authoring framework largely for HTML documents.

Portable Unix Documentation provides two mini-languages for authoring in the unix environment. These languages, pud-man and pud-faq are both written in zoem.

EXAMPLES

This is a relatively new section, aimed at assembling useful or explanatory snippets.

Create a vararg containing file names matching a pattern (`png` in this example).

```
\setx{images}{
   \inspect{
      {mods}{iter-lines,discard-miss}
   }{(.*~.png)}{_#1{{\1}}}{\system{ls}}
}
```

Use magic boundary stops with `tr#2`.

```
\tr{
   {from}{[:lower:][:upper:][:digit:][:space:][:punct:]}
   {to}{[*L#][*U#][*D#][*S#][*P#]}}{
 !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\\]^_`
abcdefghijklmnopqrstuvwxyz
\{|\}~]}
```

AUTHOR

Stijn van Dongen.