NAME

json_xs – JSON::XS commandline utility

SYNOPSIS

```
json_xs [-v] [-f inputformat] [-t outputformat]
```

DESCRIPTION

json_xs converts between some input and output formats (one of them is JSON).

The default input format is `json` and the default output format is `json-pretty`.

OPTIONS

-v Be slightly more verbose.

-f fromformat

Read a file in the given format from STDIN.

`fromformat` can be one of:

json – a json text encoded, either utf-8, utf16-be/le, utf32-be/le
cbor – CBOR (RFC 7049, CBOR::XS), a kind of binary JSON
storable – a Storable frozen value
storable-file – a Storable file (Storable has two incompatible formats)
bencode – use Convert::Bencode, if available (used by torrent files, among others)
clzf – Compress::LZF format (requires that module to be installed)
eval – evaluate the given code as (non-utf-8) Perl, basically the reverse of "-t dump"
yaml – YAML format (requires that module to be installed)
string – do not attempt to decode the file data
none – nothing is read, creates an `undef` scalar – mainly useful with `-e`

-t toformat

Write the file in the given format to STDOUT.

`toformat` can be one of:

json, json-utf-8 – json, utf-8 encoded
json-pretty – as above, but pretty-printed
json-utf-16le, json-utf-16be – little endian/big endian utf-16
json-utf-32le, json-utf-32be – little endian/big endian utf-32
cbor – CBOR (RFC 7049, CBOR::XS), a kind of binary JSON
cbor-packed – CBOR using extensions to make it smaller
storable – a Storable frozen value in network format
storable-file – a Storable file in network format (Storable has two incompatible formats)
bencode – use Convert::Bencode, if available (used by torrent files, among others)
clzf – Compress::LZF format
yaml – YAML::XS format
dump – Data::Dump
dumper – Data::Dumper
string – writes the data out as if it were a string
none – nothing gets written, mainly useful together with `-e`

Note that Data::Dumper doesn't handle self-referential data structures correctly – use "dump" instead.

-e code

Evaluate perl code after reading the data and before writing it out again – can be used to filter, create or extract data. The data that has been written is in `$_`, and whatever is in there is written out afterwards.

EXAMPLES

```
json_xs -t none <isitreally.json
```

"JSON Lint" – tries to parse the file *isitreally.json* as JSON – if it is valid JSON, the command outputs

nothing, otherwise it will print an error message and exit with non-zero exit status.

```
<src.json json_xs >pretty.json
```

Prettify the JSON file *src.json* to *dst.json*.

```
json_xs -f storable-file <file
```

Read the serialised Storable file *file* and print a human-readable JSON version of it to STDOUT.

```
json_xs -f storable-file -t yaml <file
```

Same as above, but write YAML instead (not using JSON at all :)

```
json_xs -f none -e '$_ = [1, 2, 3]'
```

Dump the perl array as UTF-8 encoded JSON text.

```
<torrentfile json_xs -f bencode -e '$_ = join "\n", map @$_, @{$_->{"announce-
```

Print the tracker list inside a torrent file.

```
lwp-request http://cpantesters.perl.org/show/JSON-XS.json | json_xs
```

Fetch the cpan-testers result summary `JSON::XS` and pretty-print it.

AUTHOR