Package 'msgpack'

February 22, 2018

Title A Compact, High Speed Data Format

Version 1.0

Description A fast C-based encoder and streaming decoder for the 'messagepack' data format. 'Messagepack' is similar in structure to 'JSON' but uses a more compact binary encoding. Based on the CWPack C library.

Depends R (>= 3.3.0)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Suggests testthat, R.rsp

VignetteBuilder R.rsp

RoxygenNote 6.0.1

NeedsCompilation yes

Author Peter Meilstrup [aut, cre, cph],
Claes Wihlborg [ctr, cph],
Björn Höhrmann [ctr, cph]

Maintainer Peter Meilstrup <peter.meilstrup@gmail.com>

Repository CRAN

Date/Publication 2018-02-22 09:15:35 UTC

R topics documented:

msgConnection 2
packMsg 4
unpackMsg 5

Index **8**

msgConnection *Read and write msgpack formatted messages over R connections.*

Description

A msgConnection object decodes msgpack messages from an underlying R raw connection.

readMsg(con) reads exactly one message from a msgConnection, or throws an error.

writeMsg(x, con) writes a single message to a msgConnection.

writeMsg will work with any R connection in raw mode, but reading requires a msgConnection object.

Usage

```
msgConnection(con, read_size = 2^16, max_size = NA, ...)

## S3 method for class 'msgConnection'
close(con, ...)

partial(con)

## S3 method for class 'msgConnection'
partial(con)

readMsgs(con, n = NA, ...)

status(con)

## S3 method for class 'msgConnection'
status(con)

## S3 method for class 'msgConnection'
seek(con, rw = summary(con)$mode, ...)

readMsg(con, ...)

writeMsg(obj, con, ...)

writeMsgs(objs, con, ...)
```

Arguments

con	A connection object open in binary mode.
read_size	How many bytes to read at a time.
max_size	The largest partial message to store, in bytes. NA means do not enforce a limit.
...	Unpacking options (see unpackMsg).

n	The maximum number of messages to read. A value of NA means to parse all available messages until end of input.
rw	See seek().
obj	An R object.
objs	A list of R objects.

Details

Because msgpack messages have unpredictable length, the decoder reads ahead in chunks, then finds the boundaries between messages. Therefore when reading over a socket or a fifo it is best to use a nonblocking connection, and it will not work to mix readMsg and readBin on the same connection.

If you are reading data from a not completely trusted source you should specify options max_size and max_depth (see unpackOpts). Without it, some deeply nested or cleverly designed messages can cause a stack overflow or out-of-memory error. With these options set, you will get an R exception instead.

Value

msgConnection() returns an object of class msgConnection.

partial(con) returns any data that has been read ahead of the last decoded message.

readMsgs(con, n) returns a list of up to n decoded messages.

status(con) returns the status of msgpack decoding on the connection. A value of "ok" indicates all requested messages were read, "buffer underflow" for a non-blocking connection indicates that only part of a message has been received, and "end of input" means the last available message has been read. Other values indicate errors encountered in decoding, which will effectively halt reading.

seek(con) returns the number of bytes that have been successfully read or written, depending on the mode of the connection. (Repositioning is not supported.)

readMsg(con) returns one decoded message.

Examples

```
out <- rawConnection(raw(0), open="wb")
apply(quakes, 1, function(x) writeMsg(x, out))
length(rawConnectionValue(out))
inn <- msgConnection(rawConnection(rawConnectionValue(out), open="rb"))
readMsg(inn)
readMsgs(inn, 3)
```

packMsg *Convert R objects to msgpack format.*

Description

Convert R objects to msgpack format.

`packOpts()` interprets the `...` argument of packMsg and packMsgs. it is not exported.

Usage

```
packMsg(x, ...)

packMsgs(xs, ...)

packOpts(compatible = FALSE, as_is = FALSE, use_dict = TRUE,
  max_size = NA, buf_size = 512)

prepack(x)

## Default S3 method:
prepack(x)

## S3 method for class 'data.frame'
prepack(x)
```

Arguments

`x`	An R object, which can be null, a vector, list, environment, raw, or any combinations thereof.
`...`	Options passed to `packOpts()`
`xs`	a list of objects to pack.
`compatible`	If TRUE, emitted bytes conform to version 1.0 of msgpack encoding. This means that msgpack strings are used for raw objects.
`as_is`	If TRUE, singletons (R primitive vectors of length 1 having no names attribute) are encoded as msgpack arrays of length 1. Otherwise singletons are simplified to msgpack scalars.
`use_dict`	If TRUE, vectors having a "names" attribute are encoded as dicts. If false, they are encoded as arrays and the names are discarded.
`max_size`	The largest buffer that will be allocated.
`buf_size`	The initial amount of memory, in bytes, to allocate for packing each message. This will be dynamically grown if a larger message is passed, so there is little reason to change this.

Details

Strings are re-encoded to UTF-8 if necessary. Real numbers taking integral values may be emitted as integers to save space.

Normally an R vector of length 1 will be unboxed, e.g. packMsg(1) will make a msgpack integer, but packMsg(c(1,2)) will make a msgpack list. To prevent this and produce a list of length 1 in the first case, specify `as_is = TRUE`. Objects of class `AsIs` or `data.frame` will always be encoded as-is.

A hook for pre-processing R objects before packing is supported, by giving the object an S3 class and implementing a method `prepack`. For instance, `prepack.data.frame(x)` simply adds the `"AsIs"` class to `x`.

Environment objects are written out with the keys in sorted order, but named vectors are written in the order which the entries appear.

Object attributes other than `names` and `class` are ignored.

Value

An object of class "raw".

Examples

```
packMsg( list(compact=TRUE, schema=0) )
x <- packMsgs(list("one", "two", "three"))
unpackMsgs(x, 2)
```

---

unpackMsg *Decode msgpack messages.*

Description

`unpackMsg` converts a raw array containing one message in msgpack format into the corresponding R data structure.

`unpackMsgs` extracts a number of msgpack messages from a raw object.

`unpackOpts()` interprets is passed to `...` in `unpackMsgs()`, `unpackMsg()`, and `msgConnection()`. It is not exported.

Usage

```
unpackMsg(x, ...)

unpackMsgs(x, n = NA, reader = NULL, ...)

unpackOpts(parent = NULL, df = TRUE, simplify = TRUE, max_size = NA,
  max_depth = NA, underflow_handler = NULL)
```

Arguments

x	A raw() object, perhaps read from a file or socket.
...	Options passed to unpackOpts.
n	How many messages to read. An "NA" here means to read as much as possible.
reader	For implementing connections; a function that takes no arguments and returns a raw containing more data.
parent	When an environment is given, (such as emptyenv()), unpack msgpack dicts into environment objects, with the given value as parent. This option overrides use_df=TRUE. Otherwise, unpack dicts into named vectors / lists.
df	When TRUE, convert msgpack dicts, whose elements are all arrays having the same length, into data.frame()s.
simplify	If TRUE, simplify msgpack lists into primitive vectors.
max_size	The maximum length of message to decode.
max_depth	The maximum degree of nesting to support.
underflow_handler	Used internally.

Details

The msgpack format does not have typed arrays, so all msgpack arrays are effectively lists from the R perspective. However, if an array containing compatibly typed elements is read, unpack will return a logical, integer, real or string vector as appropriate. This behavior is disabled with simplify=FALSE. The coercion used is more conservative than R's coercion: Integer values may be converted to real, but boolean values will not be cast to numeric, nor any types to string. If conversion from a large integer to real loses precision, a warning is printed.

Msgpack also does not distinguish between NA and NULL. All nils will be decoded as NA.

Strings are assumed to be UTF-8 encoded. If a msgpack string does not appear to be valid UTF-8, a warning is printed and a raw object is produced instead.

Msgpack allows any type to be the key of a dict, but R only supports strings. If a non-string appears as key in a msgpack dict, it will be converted to string with deparse().

Extension types will be decoded as raw objects with a class like "ext120" and a warning.

Value

unpackMsg(x) returns one decoded message (which might be shorter than the input raw), or throws an error.

unpackMsgs(r, n) returns a list X with four elements:

- X$msgs is a list of the messages unpacked.
- X$remaining is data remaining to be parsed.
- X$status is a status message, typically "ok", "end of input", or "buffer underflow".
- x$bytes_read the number of bytes consumed.

Examples

```
msg <- as.raw(c(0x82, 0xa7, 0x63, 0x6f, 0x6d, 0x70, 0x61, 0x63, 0x74, 0xc3,
                0xa6, 0x73, 0x63, 0x68, 0x65, 0x6d, 0x61, 0x00))
unpackMsg(msg)
x <- packMsgs(list("one", "two", "three"))
unpackMsgs(x, 2)
```

Index

class, *5*
close.msgConnection *(*msgConnection*)*, 2
connection, *2*

data.frame(), *6*
deparse(), *6*

emptyenv(), *6*

msgConnection, 2
msgConnection(), *5*

packMsg, 4
packMsgs *(*packMsg*)*, 4
packOpts *(*packMsg*)*, 4
packOpts(), *4*
partial *(*msgConnection*)*, 2
prepack *(*packMsg*)*, 4

raw(), *6*
readMsg *(*msgConnection*)*, 2
readMsgs *(*msgConnection*)*, 2

seek(), *3*
seek.msgConnection *(*msgConnection*)*, 2
status *(*msgConnection*)*, 2

unpackMsg, *2*, 5
unpackMsg(), *5*
unpackMsgs *(*unpackMsg*)*, 5
unpackMsgs(), *5*
unpackOpts, *3*, *6*
unpackOpts *(*unpackMsg*)*, 5
unpackOpts(), *5*

writeMsg *(*msgConnection*)*, 2
writeMsgs *(*msgConnection*)*, 2