Package: zumime (via r-universe)

November 15, 2026

Title Media Type Detection from Bytes and File Names

Version 0.1.0.9000

Description Tells what a file is from its first bytes and from its name, with no system library. Implements the Web Hypertext Application Technology Working Group (WHATWG) Multipurpose Internet Mail Extensions (MIME) Sniffing Standard <https://mimesniff.spec.whatwg.org/> over raw vectors and paths. Reads inside containers to tell 'Office' documents from plain 'ZIP' archives, audio from video in 'MP4' files, and 'Word' from 'Excel' in legacy 'Office' files, and recognises the data files 'R' users hold, such as files saved by 'R', 'Parquet', 'SQLite' and 'Stata' files. Every type has a parent in a hierarchy, and an extension table generated from the Internet Assigned Numbers Authority (IANA) media types registry and the 'Apache' 'httpd' type map answers from the name. One call reports both answers, whether they agree, and whether a file is compressed, encrypted or truncated.

License MIT + file LICENSE

URL https://github.com/pedrobtz/zumime, https://pedrobtz.github.io/zumime/

BugReports https://github.com/pedrobtz/zumime/issues

Depends R (>= 4.1)

Suggests testthat (>= 3.0.0), withr, zucsv, zujson, zukomp

Config/Needs/website knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Repository https://pedrobtz.r-universe.dev

Date/Publication 2026-11-15 06:29:11 UTC

RemoteUrl https://github.com/pedrobtz/zumime

RemoteRef HEAD

RemoteSha fae69af034ad2ef7d632ee0aae31a4ddd91d1a2d

Contents

mime_check 2
mime_extensions 3
mime_guess 4
mime_is 5
mime_parent 5
mime_parse 6
mime_register 7
mime_sniff 8
mime_sniffer 10
mime_type 11
mime_types 13
zumime_info 14

Index **15**

`mime_check` *Is a file one of these types?*

Description

The upload question: `TRUE` when the bytes say the file is one of `allowed`, or a descendant of one, at `min_confidence` or better. The name is never consulted, since the question exists because names lie.

Usage

```
mime_check(x, allowed, min_confidence = "probable")
```

Arguments

`x`	A raw vector, or a character vector of paths.
`allowed`	A character vector of types, aliases or top-level types, as `mime_is()` takes them.
`min_confidence`	The weakest evidence accepted. `"probable"`, the default, accepts byte evidence of either kind. `"definite"` is for allowlists that must not trust a text heuristic. `"extension"` and `"none"` are accepted for completeness.

Details

The answer always equals `mime_is(t$type, allowed) & t$confidence >= min_confidence`, with `t <- mime_type(x)`, or `mime_sniff(x)` for a raw vector. It is computed in one pass, with a shortcut that does not change the answer. The container readers, text readers and tail reads are skipped for every input whose first answer could not end as an allowed type. No result frame is built. Call `mime_type()` to learn why a file was refused.

Value

A logical vector, one element per path (one for a raw vector); `FALSE` for an unreadable path unless `min_confidence` is `"none"`, where it is `NA`.

See Also

`mime_type()` for the full answer.

Examples

```
dir <- tempfile()
dir.create(dir)
png <- as.raw(c(0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a))
writeBin(png, file.path(dir, "photo.pdf"))
writeLines("%PDF-1.7", file.path(dir, "real.pdf"))
paths <- file.path(dir, c("photo.pdf", "real.pdf"))
mime_check(paths, c("application/pdf", "image/jpeg"))
mime_check(png, "image")
unlink(dir, recursive = TRUE)
```

`mime_extensions` *Extensions of a type*

Description

Extensions of a type

Usage

```
mime_extensions(type)
```

Arguments

`type`	A character vector of types. Aliases, parameters and case are ignored.

Value

A list with one character vector per type, the canonical extension first, without the dot. An unknown type gives `character(0)`.

See Also

mime_guess() for the reverse.

Other registry: mime_guess(), mime_is(), mime_parent(), mime_types()

Examples

```
mime_extensions(c("image/jpeg", "application/x-sqlite3"))
```

---

mime_guess *Guess a type from a file name*

Description

Takes the longest known suffix after the last directory separator, ignoring case, so .tar.gz is known as a compound and report.FINAL.PDF is a PDF. Nothing is read from the file.

Usage

```
mime_guess(path)
```

Arguments

path	A character vector of paths or file names.

Value

A character vector of canonical types, NA where no suffix is known.

See Also

mime_extensions() for the reverse.

Other registry: mime_extensions(), mime_is(), mime_parent(), mime_types()

Examples

```
mime_guess(c("report.FINAL.PDF", "data.tar.gz", "README", "x.rds"))
```

mime_is *Is a type one of these?*

Description

TRUE when a type equals one of what, descends from it, or has it as its top-level type ("image"). Aliases, parameters and case are ignored on both sides. An unknown type descends from application/octet-stream only.

Usage

```
mime_is(x, what)
```

Arguments

x	A character vector of types.
what	A character vector of types, aliases or top-level types.

Value

A logical vector the length of x, NA where x is NA.

See Also

mime_parent() for the lineage.

Other registry: mime_extensions(), mime_guess(), mime_parent(), mime_types()

Examples

```
mime_is("application/x-sqlite3", "application/vnd.sqlite3")
mime_is(c("image/png", "text/csv"), "image")
mime_is("image/svg+xml", "application/xml")
```

mime_parent *The lineage of a type*

Description

Every type has one parent, ending at application/octet-stream: text types descend from text/plain, XML dialects from application/xml, JSON dialects from application/json, ZIP-based formats from application/zip.

Usage

```
mime_parent(type)
```

Arguments

type	A character vector of types. Aliases, parameters and case are ignored.

Value

A list with one character vector per type: its ancestors, nearest first, empty for the root and NA for an unknown type.

See Also

mime_is() to test membership.

Other registry: mime_extensions(), mime_guess(), mime_is(), mime_types()

Examples

```
mime_parent("image/svg+xml")
mime_parent(c("application/x-sqlite3", "application/octet-stream"))
```

---

mime_parse *Parse and serialise MIME types*

Description

mime_parse() is the WHATWG MIME Sniffing Standard's "parse a MIME type". Type and subtype are lower-cased HTTP tokens. Parameters keep their order, and their names are lower-cased. Quoted values are unescaped. A repeated name keeps its first value. mime_format() is "serialize a MIME type". It quotes a value that is empty or not a token.

Usage

```
mime_parse(x, strict = FALSE)

mime_format(record)
```

Arguments

x	A character vector of MIME types, such as Content-Type header values.
strict	If TRUE, an input that fails to parse raises zumime_parse_error instead of giving NULL.
record	One record from mime_parse(), or a list of them.

Details

mime_format() checks a record before serialising it, so an edited record cannot carry a line break into a header. Type, subtype and parameter names must be non-empty HTTP tokens. Values must hold only HTTP quoted-string token code points: tab, U+0020 to U+007E and U+0080 to U+00FF. A record that breaks this raises zumime_type_error.

Value

`mime_parse()` returns a list with one record per input: a list of `type`, `subtype` and `parameters` (a named list of strings), or `NULL` for input that fails to parse and for `NA`.

`mime_format()` returns a character vector, `NA` for a `NULL` record.

References

https://mimesniff.spec.whatwg.org/#parsing-a-mime-type

See Also

`mime_sniff()`, which takes a supplied type in this grammar.

Examples

```
rec <- mime_parse("Text/HTML; Charset=\"UTF-8\"; charset=latin1")
str(rec)
mime_format(rec)
mime_format(mime_parse("text/plain;name=\"a b\""))
mime_parse("not a type")
```

---

mime_register *Register a type for the session*

Description

Adds a type under a parent in the hierarchy, so that `mime_sniff()`, `mime_type()`, `mime_check()` and the registry functions know it until the session ends or `mime_unregister()` removes it. A package that wants its format recognised registers it in its `.onLoad()`. It unregisters it in its `.onUnload()`, since registering a type that exists is an error.

Usage

```
mime_register(type, parent, extensions = NULL, rule = NULL)

mime_unregister(type)
```

Arguments

`type`	The new type, such as `"application/x-myformat"`.
`parent`	An existing type the new one descends from.
`extensions`	File extensions for `mime_guess()`, without the dot, or `NULL`.
`rule`	`NULL` for a type known by name only; a list of `offset` (a whole number), `bytes` (a raw vector or a string, at most 64 bytes) and an optional `mask` (a raw vector as long as `bytes`); or a function of one raw vector, the bytes read.

Details

A data rule, `list(offset, bytes, mask)`, runs in C whenever the bytes have been typed as the parent. It costs what a built-in signature costs. It is reported at `"definite"`. It sees only the bytes read: a rule that ends past 4096 bytes needs a larger `limit`. A function rule is called from R with the bytes read and must return `TRUE` or `FALSE`. It runs last, only when the parent matched, and is reported at `"probable"`. A function rule that raises an error counts as `FALSE`, and the call warns once with a `zumime_rule_warning` naming the type. A data rule cannot sit under a type that a function rule decides, since data rules run first.

Both kinds refine only what zumime sniffs in the `"none"` context. They never run in another context, nor on a supplied type that stands.

Value

`mime_register()` returns the type, invisibly. `mime_unregister()` returns `NULL`, invisibly.

See Also

`mime_types()`, where registered types show `detected_by = "registered"`.

Examples

```
mime_register(
  "application/x-zumime-demo",
  "application/octet-stream",
  extensions = "zdemo",
  rule = list(offset = 0, bytes = "ZDEMO")
)
mime_sniff(charToRaw("ZDEMO\x01\x02"))
mime_guess("file.zdemo")
mime_unregister("application/x-zumime-demo")
```

---

mime_sniff *Sniff a type from bytes*

Description

Runs the WHATWG MIME Sniffing Standard's "MIME type sniffing algorithm" over the first bytes of a raw vector, with its three inputs: the supplied type (a `Content-Type` value, or `NULL`), the no-sniff flag, and the sniffing context.

Usage

```
mime_sniff(
  x,
  type = NULL,
  no_sniff = FALSE,
  context = c("none", "browsing", "image", "audio_video", "font"),
  limit = 4096L
)
```

Arguments

x	A raw vector, or a character vector of paths, which are read as mime_type() reads them; an unreadable path gives a row with type NA.
type	The supplied type, such as a Content-Type header value, or NULL when there is none. A value that does not parse as a MIME type is treated as NULL.
no_sniff	TRUE for the no-sniff flag (X-Content-Type-Options: nosniff): a supplied type is then returned as is.
context	One of "none", "browsing", "image", "audio_video" or "font".
limit	The number of bytes to read; 0 reads them all. A raw vector longer than limit is sniffed on its first limit bytes.

Details

The "browsing" context is what a browser computes for a navigation: the standard's tables and nothing else. The default "none" asks the plain file question: the same algorithm, then zumime's own signatures (compression wrappers, tar, ISO 9660, RAR 5, 7z, fonts, WebAssembly) where the standard found only binary bytes. "image", "audio_video" and "font" are the standard's context-specific algorithms: they look only for types of that kind and otherwise return the supplied type.

A text result carries charset, from the first rule that answers: the supplied type's charset parameter, a byte-order mark, an XML declaration, an HTML <meta> prescan, valid UTF-8, then the C1 rule. The C1 rule gives windows-1252 when a byte in 0x80 to 0x9F appears, else ISO-8859-1. It is a guess, so it lowers confidence to "probable".

Value

A zumime_result data frame with one row per path (one for a raw vector) and columns path (NA for a raw vector), type (a canonical type), confidence (an ordered factor: none < extension < probable < definite), charset, inner, encrypted, complete, newline ("lf", "crlf", "cr", "mixed", or NA) and interpreter. inner and complete are always NA here; mime_type() fills them.

References

https://mimesniff.spec.whatwg.org/

See Also

mime_parse() for the supplied type's grammar.

Examples

```
png <- as.raw(c(0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a))
mime_sniff(png)
mime_sniff(charToRaw("<!DOCTYPE html><title>x</title>"))

# A server that says text/plain about a binary file
mime_sniff(png, type = "text/plain")
```

```
# The supplied type wins under nosniff
mime_sniff(png, type = "text/csv", no_sniff = TRUE)
```

`mime_sniffer` *Sniff a stream chunk by chunk*

Description

`mime_sniffer()` starts an incremental sniffer; `mime_push()` hands it the next chunk and returns `NULL` while the bytes seen cannot yet decide the type, or the result row once they can. `mime_finish()` forces an answer from what has been seen, as the end of a short stream would. The answer is the one `mime_sniff()` gives for the whole stream with the same `limit`. A stream exactly `limit` bytes long is read as one that goes on, since the sniffer cannot know it has ended, so a character cut at the end does not count against UTF-8.

Usage

```
mime_sniffer(
  type = NULL,
  no_sniff = FALSE,
  context = c("none", "browsing", "image", "audio_video", "font"),
  limit = 4096L
)

mime_push(s, chunk)

mime_finish(s)
```

Arguments

`type`	The supplied type, such as a `Content-Type` header value, or `NULL` when there is none. A value that does not parse as a MIME type is treated as `NULL`.
`no_sniff`	`TRUE` for the no-sniff flag (`X-Content-Type-Options: nosniff`): a supplied type is then returned as is.
`context`	One of `"none"`, `"browsing"`, `"image"`, `"audio_video"` or `"font"`.
`limit`	The number of bytes to read; `0` reads them all. A raw vector longer than `limit` is sniffed on its first `limit` bytes.
`s`	A sniffer from `mime_sniffer()`.
`chunk`	A raw vector, the next bytes of the stream.

Details

Once the sniffer has decided, further pushes ignore their chunks and return the same row. Once it has finished, a push raises a `zumime_state_error`; `mime_finish()` can still be called and returns the same row.

The bytes decide as soon as one of three things holds. No longer signature could still override the match. The container reader for the type has what it needs. Or `limit` bytes have arrived. Text decides only at `limit` or at the finish, since a later byte can still change it. The tail-reading column, `complete`, is `NA` for a stream.

The sniffer keeps its bytes in an R raw vector, grown as they arrive up to `limit`. With `limit = 0` text never decides before the finish, so the sniffer holds the whole stream until `mime_finish()`. A sniffer saved and loaded again keeps those bytes but forgets its decision, and is undecided until the next push.

Value

`mime_sniffer()` returns a `zumime_sniffer`. `mime_push()` returns `NULL` or a one-row `zumime_result`; `mime_finish()` always returns the result.

See Also

`mime_sniff()` for bytes in hand.

Examples

```
s <- mime_sniffer()
mime_push(s, charToRaw("GIF8"))
mime_push(s, charToRaw("9a"))

# Text waits for the limit or the finish.
t <- mime_sniffer(limit = 64)
mime_push(t, charToRaw("hello, "))
mime_finish(t)
```

---

mime_type *Type files from their bytes and their names*

Description

Reads the first bytes of each path in one C loop, types them as `mime_sniff()` does, guesses a type from the name as `mime_guess()` does, and says whether the two agree. The other columns answer what a consumer asks before opening a file: what a compressed file holds, whether it is encrypted or truncated, its line endings, and a script's interpreter.

Usage

```
mime_type(
  path,
  limit = 4096L,
  complete = FALSE,
  inner = c("name", "decompress"),
  text = c("whatwg", "family"),
  charset = c("declared", "detect"),
```

```
  strict = FALSE
)
```

Arguments

path	A character vector of paths.
limit	The number of bytes to read from each file; 0 reads whole files.
complete	If TRUE, also read each file's end and fill complete. A ZIP needs its end-of-central-directory record. A PDF needs %%EOF in its last kilobyte. A gzip stream must inflate to its trailer. Parquet must end in PAR1. Other formats give NA. A PDF is encrypted when its trailer dictionary, found in the last 4 KB, names /Encrypt.
inner	How to find the type inside a compression wrapper (gzip, bzip2, xz, zstd, LZ4, lzip, compress): "name" takes it from the rest of the name, so data.csv.gz holds text/csv; "decompress" decompresses gzip with zukomp and sniffs the result, falling back to the name for other wrappers and for files over 16 MiB.
text	"whatwg" keeps the standard's answer for text; "family" offers a still generic text/plain to the family's sniffers, which read whole files.
charset	"declared" reports the charset the bytes declare; "detect" asks zuutf where they declare none, or only the C1 guess.
strict	If TRUE, an unreadable path raises zumime_io_error.

Details

Unreadable and missing paths give a row with type NA and confidence "none", so one bad entry does not abort a scan of a tree. strict = TRUE raises zumime_io_error instead. A directory is inode/directory, and nothing is read from it.

agree is TRUE when the byte type and the name type lie on one lineage, so bytes that say text/plain agree with a .csv name and a ZIP agrees with a .docx name; FALSE when they differ, such as a PNG named .txt; and NA when either is unknown. A disagreement is a finding, not an error.

The default answer depends on nothing outside the package. Three arguments ask for an optional family package. inner = "decompress" uses zukomp and charset = "detect" uses zuutf; either raises zumime_dependency_error when its package is absent. text = "family" never raises: it uses whichever of zucsv, zujson and zutoml are installed, listed by zumime_info()$family, and with none it keeps the standard's answer. zuutf and zutoml are not yet on CRAN; install them from GitHub. An error inside an optional package leaves that answer as it was, so a corrupt stream keeps the name's inner.

Value

A zumime_result data frame with one row per path and columns path, type, confidence, charset, by_name, agree, inner, encrypted, complete, newline and interpreter.

See Also

mime_sniff() for bytes and the standard's contexts, mime_check() for an allowlist.

Examples

```
dir <- tempfile()
dir.create(dir)
writeLines(c("a,b", "1,2"), file.path(dir, "data.csv"))
saveRDS(mtcars, file.path(dir, "cars.rds"))
writeBin(
  as.raw(c(0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a)),
  file.path(dir, "logo.txt")
)
mime_type(list.files(dir, full.names = TRUE))
unlink(dir, recursive = TRUE)
```

---

mime_types *The type registry*

Description

Every type zumime knows, with its parent, aliases, extensions and the kind of evidence that detects it.

Usage

```
mime_types()
```

Value

A data frame with one row per canonical type and columns `type`, `parent` (`NA` for the root, `application/octet-stream`), `aliases` and `extensions` (list columns of character vectors, the canonical extension first), `category` (the top-level type) and `detected_by` (`"signature"`, `"container"`, `"text"`, `"extension"` or `"registered"`).

See Also

`mime_parent()` for one type's lineage.

Other registry: `mime_extensions()`, `mime_guess()`, `mime_is()`, `mime_parent()`

Examples

```
types <- mime_types()
nrow(types)
types[types$type == "application/zip", ]
```

zumime_info *Build information*

Description

Reports what this build of zumime contains: the package version and the version of the generated type tables, which is the date of the upstream sources they were built from.

Usage

```
zumime_info()
```

Value

A list with `version`, the package version as a string; `table_version`, the date of the table sources as a string (`"YYYY-MM-DD"`); and `family`, the optional family packages this session can offer text to with `mime_type(text = "family")`.

See Also

`mime_type()` for the arguments that use the family.

Examples

```
zumime_info()
```

Index

∗ **registry**
    mime_extensions, 3
    mime_guess, 4
    mime_is, 5
    mime_parent, 5
    mime_types, 13

mime_check, 2
mime_check(), *7*, *12*
mime_extensions, 3
mime_extensions(), *4–6*, *13*
mime_finish *(*mime_sniffer*)*, 10
mime_format *(*mime_parse*)*, 6
mime_guess, 4
mime_guess(), *4–7*, *11*, *13*
mime_is, 5
mime_is(), *2*, *4*, *6*, *13*
mime_parent, 5
mime_parent(), *4*, *5*, *13*
mime_parse, 6
mime_parse(), *9*
mime_push *(*mime_sniffer*)*, 10
mime_register, 7
mime_sniff, 8
mime_sniff(), *7*, *10–12*
mime_sniffer, 10
mime_type, 11
mime_type(), *3*, *7*, *9*, *14*
mime_types, 13
mime_types(), *4–6*, *8*
mime_unregister *(*mime_register*)*, 7
mime_unregister(), *7*

zumime_info, 14