Mahjong Portable Format

MJPF

An open, platform-neutral, **additive** specification for sharing a Mahjong deal, a continuable game, or a played result — with no server, no account, and no network.

Proposed, originated & first specified by JSCBIZ

White paper & normative overview

Specification version 1.2.0 · Status: Stable

File extension `.mjpf` (JSON) · Media type `application/vnd.mahjong.mjpf+json`

© 2026 JSCBIZ. This document is published openly so that any Mahjong application may implement and interoperate with the format free of charge and free of royalty. The Mahjong Portable Format was proposed and first specified by JSCBIZ; see §“Attribution” and §“Licence”.

Canonical version & downloads: https://mjpf.openmahj.org

Contents

© 2026 JSCBIZ. Open specification — free to implement (see §License).

© 2026 JSCBIZ. Open specification — free to implement (see §License).

1. Abstract & motivation

The **Mahjong Portable Format (MJPF)** is a small, self-describing, platform-neutral document for sharing a Mahjong **deal**, a continuable **resume** bundle, or a played **result**, so that two people can compete on the *exact same deterministic deal* without any backend, account, or network connection.

The insight is that a well-built Mahjong engine is **deterministic**: the entire game is a pure function of a single random seed and a rule set. You therefore never need to transmit the game — you transmit the seed. Encode a seed and a rule-set identifier in a tiny QR code (or a short copyable text code), and any conforming reader reconstructs the identical wall and plays the identical hand, entirely offline. When both players finish, they compare results by eye or by passing a one-line result code back.

1.1 Design goals

- **Backend-free competition.** A challenge is a deterministic deal; anyone who can reproduce it plays the identical hand and can compare results — no server, no account, no network.
- **Tiny and offline.** A bare deal fits in a single QR code and everything works fully offline.
- **Platform-neutral.** JSON over UTF-8; no language- or vendor-specific serialization in the portable core; 64-bit integers that exceed the JSON safe range are carried as strings.
- **Self-describing & versioned.** Every document declares its `format`, its semantic-version `spec`, and the capabilities it requires, so a reader can decide safely whether it can act on it.
- **Forward-compatible.** Old readers tolerate documents from newer minor versions; a document is refused only when it requires a capability the reader does not understand (or declares a newer major version).
- **Extensible without forking.** A reserved vendor-extension namespace lets other apps attach data that everyone else ignores gracefully.

This white paper is a readable companion to the normative specification. Where this document and the machine-readable specification/JSON Schema differ, the normative specification and schema govern. The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** are to be interpreted as described in RFC 2119.

2. The format meta-layout

An MJPF document is a JSON object. Two equivalent serializations exist: the canonical, human-readable **JSON profile** (`.mjpf.json`), and the **MJQR transport** (§5), which is the same document compressed for QR codes, links, and copy-paste. A reader MUST ignore any top-level field it does not recognize.

Field	Type	Req.	Meaning
`format`	string	yes	Magic discriminator. MUST be “mjpf”.
`spec`	string	yes	Spec version, MAJOR.MINOR.PATCH (semver).
`kind`	string	yes	“deal”, “resume”, or “result” (§3).

© 2026 JSCBIZ. Open specification — free to implement (see §License).

Field	Type	Req.	Meaning
`requires`	string[]	yes	Capability flags a reader MUST understand to use the document (§3.4). MAY be empty.
`uses`	string[]	yes	Advisory capability flags. A reader MAY ignore unknown entries. MAY be empty.
`game`	object	yes	The deterministic game reference (§2.1).
`events`	array	no	Canonical move-by-move event log; present only with the event-log capability (§3.2).
`result`	object	no	A played outcome (§2.2); present for kind:“result” or a deal carrying a score-to-beat.
`meta`	object	no	Descriptive metadata (§2.3). Never required to reconstruct a game.
`ext`	object	no	Vendor-extension namespace (§3.5).

Table 1 — Top-level document fields.

2.1 game — the deterministic reference

The `game` object is the platform-neutral core: it is everything a reader needs to reconstruct the deal.

Field	Type	Meaning
`seed`	string	64-bit RNG seed as a decimal string (§5.2).
`ruleset`	string	Rule-set id: simplified, hongkong, riichi-lite, mcr, taiwanese16.
`tier`	string	AI opponent tier: basic, moderate, veryGood, wizard.
`mode`	string	training or play.
`seats`	int	Number of seats (4 for the standard game).
`humanSeat`	int	The seat the importer plays (0–3).
`recordSeat`	int	The seat whose recorded decisions drive the shared past. Equals humanSeat unless a branch.
`branchPointIndex`	int	Event index at which control switches seats (0 if not a branch).

Table 2 — The game reference. A branch (“take over a seat”) is denoted by humanSeat ≠ recordSeat and branchPointIndex > 0, and requires the branch capability.

2.2 result — a played outcome

Field	Type	Req.	Meaning
`won`	bool	yes	Did the player win the hand?
`score`	int	yes	Points in the rule set's own unit (faan/han/fan/tai → resolved points).
`bySelfDraw`	bool	no	Win by self-draw vs. discard.
`turns`	int	no	Turns taken.
`label`	string	no	Human-readable summary, e.g. “Riichi · 7700”.

Table 3 — The result object.

© 2026 JSCBIZ. Open specification — free to implement (see §License).

2.3 meta — descriptive metadata

`meta` may carry `title` (string), `createdAt` (ISO-8601 string, supplied by the producer), `app` (string), and `appVersion` (string). All are optional, and a reader MUST NOT depend on `meta` to reconstruct play.

3. Kinds & capabilities

3.1 deal — the portable competition core

A **deal** carries only `game` (and optionally a `result` as a *score-to-beat*, plus `meta`). It has **no event log**. This is the fully platform-neutral interop surface: a seed, a rule set, and a tier. A reader reconstructs a fresh, continuable game on the seeded wall, positioned at the human's first decision. `requires` is empty unless a score-to-beat is present.

3.2 resume — a continuable game

A **resume** carries the canonical `events` log so the exact paused position can be rebuilt by re-running the deterministic engine and replaying the recorded decisions. It MUST declare `requires: ["event-log"]` (and `["event-log", "branch"]` for a branch). This profile is primarily application-to-application between engines that share the event vocabulary.

3.3 result — a played outcome report

A **result** carries a `result` for a given `game` (seed + rule set), used to send "here's how I did / beat this" back to a challenger. It MUST declare `requires: ["result"]`.

3.4 Capabilities — semantic versioning of features

Capabilities are how MJPF evolves without breaking old readers. New optional features arrive in new minor versions as new capability flags. `requires[]` is the safety gate — a reader MUST understand every entry or MUST refuse the document, reporting the unknown flags. `uses[]` is advisory — a reader MAY ignore unknown entries and still act on the document.

Flag (since)	Meaning
`event-log`	The document carries a canonical move-by-move event log.
`branch`	The game uses the branch / take-over-a-seat two-segment schedule.
`result`	The document carries a played result.
`bonus-tiles`	The deal / rule set uses bonus tiles (flowers & seasons).
`view-only (1.1.0)`	Spectator playback only — no resuming, forking or analysis.
`event-vocab-2 (1.1.0)`	The event log may carry the 1.1.0 vocabulary additions.
`tournament-bundle (1.2.0)`	The document is a whole tournament — a deterministic stack of deals.

Table 4 — The capability registry through v1.2.0.

© 2026 JSCBIZ. Open specification — free to implement (see §License).

3.5 Vendor-extension namespace (ext)

`ext` is a flat JSON object reserved for app-specific data. Keys SHOULD be namespaced as `vendor.key` (e.g. `acme.note`). Readers MUST ignore `ext` keys they do not own, MUST NOT let `ext` affect game reconstruction, and MUST NOT execute anything found in it. This lets independent apps round-trip extra data through each other without coordination or a spec change.

4. The event log

When present, `events` is an ordered array of canonical game events that, replayed against a deterministic engine seeded by `game.seed`, reproduces the game byte-for-byte. The v1 event vocabulary mirrors the reference engine's canonical events:

```
dealt · drew · flowerReplaced · discarded · riichiDeclared · claimed
concealedKong · kongReplacementDraw · passed · won · exhaustiveDraw
```

v1 honesty note

The per-event JSON shape in v1 follows the reference engine's serialization (a single-key object keyed by the event name, with tiles carried by `kind` plus a stable `id`). It is fully specified by the generated example resume payload, which is the normative shape for v1. A cleaner, language-neutral event encoding is a candidate for a future minor version behind a new capability flag. The `deal` profile has no event log and is unaffected — if you implement only `deal` (the common case for cross-app competition), refuse documents that require `event-log` and you are done.

5. MJQR — the cross-platform transport

MJQR is how an MJPF document travels as a QR code, a copyable code, or a share link. It is defined as:

```
MJQR = base64( rawDEFLATE( UTF-8( canonicalJSON(document) ) ) )
```

- **canonicalJSON** — keys sorted, slashes unescaped, no insignificant whitespace (stable bytes, so a code can be hashed and compared).
- **rawDEFLATE** — raw DEFLATE, RFC 1951 (see §5.1 — the #1 interop gotcha).
- **base64** — standard base64, RFC 4648 (with +// and = padding). QR byte mode carries it directly.

A v1 reader treats ≤ **2200 UTF-8 bytes** as single-QR-safe (QR byte mode at error-correction level M holds roughly 2.3 KB). A bare `deal` always fits one QR. Larger documents (full `resume` logs) fall back to a copyable text code that the recipient pastes — the same string, no QR.

5.1 Compression is raw DEFLATE (RFC 1951), not zlib-wrapped

© 2026 JSCBIZ. Open specification — free to implement (see §License).

The one gotcha that bites everyone — read this

MJQR compression is **raw DEFLATE (RFC 1951)**, **not** zlib-wrapped (RFC 1950). The reference implementation uses Apple's `NSData.compressed(using: .zlib)`, which — despite the name — emits a **raw DEFLATE** stream with **no** 2-byte zlib header (`0x78…`) and **no** trailing Adler-32 checksum. This was verified empirically: the reference bytes begin `ab 56 …`, not `78 9c`. A non-Apple reader MUST therefore use **raw inflate**. Using the zlib-wrapped variant will fail with an "incorrect header check" error. This single rule is the most common reason a naïve port cannot read an MJQR code.

Platform	Decompress (read)	Compress (write)
Python	`zlib.decompress(data, -15)`	`compressobj(lvl, DEFLATED, -15)`
Node.js	`zlib.inflateRawSync(buf)`	`zlib.deflateRawSync(buf)`
Browser	`DecompressionStream('deflate-raw')`	`CompressionStream('deflate-raw')`
C (zlib)	`inflateInit2(&s, -15)`	`deflateInit2(&s, lvl, …, -15, 8, 0)`
Go	`flate.NewReader(r)`	`flate.NewWriter(w, lvl)`

Table 5 — Raw-DEFLATE codecs per platform. Note the -15 window-bits sign.

5.2 The 64-bit seed is a decimal string

`game.seed` is a 64-bit value but is encoded as a **decimal string**, because JSON/JavaScript numbers are only exact to 2^{53}. Readers MUST parse it as a 64-bit unsigned integer. UTF-8 throughout; field order is not semantically significant (the canonical form sorts keys only for stable bytes); no NaN/Infinity; no comments.

5.3 Transport channels

Channel	Availability	Notes
QR code (MJQR)	Cross-platform	The portable default. Any device with a camera can read it.
Copyable code (MJQR text)	Cross-platform	Universal fallback — paste into any MJPF reader. Always works.
AirDrop	Apple-only	Convenience on Apple devices; carries the same payload.
Share sheet	Apple-only	Convenience; routes the same payload to Messages/Mail/etc.

Table 6 — AirDrop and the share sheet are conveniences on Apple platforms only; they are never required. The QR code and copyable code are the cross-platform guarantees.

6. The determinism contract

Two implementations produce identical play from the same `deal` **iff** they share the same wall-construction algorithm (the seeded shuffle) and rule-set semantics. MJPF carries the seed and the rule-set **id**; it does not carry the rule-set *rules* — a reader supplies those. The reference wall algorithm is:

© 2026 JSCBIZ. Open specification — free to implement (see §License).

- **RNG — SplitMix64.** Seed the 64-bit state (if the seed is 0, use the constant 0x9E3779B97F4A7C15).
- **Tile order.** Build the canonical tile multiset for the rule set (e.g. 136 tiles for a standard set: 4× of each of the 34 suited/honor kinds; +8 bonus tiles for sets that use flowers).
- **Shuffle.** Fisher–Yates walking *front to back* (i = 0 … n–2), each step picking partner j = i + (r mod (n–i)) where the bounded index r mod (n–i) uses the **Lemire multiply-high** reduction, not rejection-modulo — see §7.4 of the spec for the exact listing. Getting this bit-identical is the whole game; a plain modulo deals a different wall for every seed.
- **Deal.** Deal opening hands in seat order, then draw from the live wall during play.

```
# SplitMix64 — the reference RNG
state = seed or 0x9E3779B97F4A7C15
def next64():
    global state
    state = (state + 0x9E3779B97F4A7C15) & MASK64
    z = state
    z = ((z ^ (z >> 30)) * 0xBF58476D1CE4E5B9) & MASK64
    z = ((z ^ (z >> 27)) * 0x94D049BB133111EB) & MASK64
    return z ^ (z >> 31)
```

Listing 1 — The reference SplitMix64 generator (matches the kit's SeededRNG).

Reproducing the wall bit-for-bit in an *independent* codebase (the exact front-to-back Fisher–Yates index derivation is the fiddly part, §7.4) is no longer left as an exercise: a from-spec reference implementation and a frozen conformance corpus ship with the format, and a standing check re-derives the wall from the spec alone and asserts it matches the vectors seed-for-seed. The §11 conformance vectors (canonicalJSON, mjqr, and per-seed wallPrefix) and that independent implementation are the ground truth.

7. Worked examples

7.1 A bare deal

© 2026 JSCBIZ. Open specification — free to implement (see §License).

```
{
  "format" : "mjpf",
  "game" : {
    "branchPointIndex" : 0,
    "humanSeat" : 0,
    "mode" : "play",
    "recordSeat" : 0,
    "ruleset" : "riichi-lite",
    "seats" : 4,
    "seed" : "12648430",
    "tier" : "wizard"
  },
  "kind" : "deal",
  "meta" : {
    "app" : "Mahjong Coach",
    "appVersion" : "1.0",
    "createdAt" : "2026-06-29T00:00:00Z",
    "title" : "Riichi deal challenge"
  },
  "requires" : [

  ],
  "spec" : "1.1.0",
  "uses" : [

  ]
}
```


Listing 2 — A bare deal (kind:“deal”): seed + rule set + tier. Empty requires ⇒ any v1 reader can play it.

7.2 The same deal as an MJQR code


```
PY9LT8QwDIT/SuVzi0KpKugNceKAhABxAHEwibfJkkdxUvFY7X/H6UpIOSSeycznA+wSBywwQdgvO2
hhxkAwHeCdMWp7n1wst9HQN0yqBbsGjI9U/fIKyYgVFo8/8pFJJzb/Iq+eMtVgdk5b13lXSGxZDBmm
od7IiHzej8PlcKFEK45YJl/uF9nAsYUPF6vFEHqRAxWsaLgsMrxDu09xbm4SaiuqTJ+Js0uxhp7VPM
1SRua6UvSqHzs1dv3Vk1LTdl62yuLrEg8bZFObGm3Re4ozVQSmz9UxCfLrmzAvpLf4U8GaT8LxDw==
```


Listing 3 — base64(rawDEFLATE(canonicalJSON(deal))). This string is the QR payload and the copyable code — both carry these identical bytes.

7.3 A challenge (deal + score-to-beat)

© 2026 JSCBIZ. Open specification — free to implement (see §License).

```
{
  "format" : "mjpf",
  "game" : {
    "branchPointIndex" : 0,
    "humanSeat" : 0,
    "mode" : "play",
    "recordSeat" : 0,
    "ruleset" : "hongkong",
    "seats" : 4,
    "seed" : "12648430",
    "tier" : "veryGood"
  },
  "kind" : "deal",
  "meta" : {
    "app" : "Mahjong Coach",
    "title" : "Beat my score"
  },
  "requires" : [

  ],
  "result" : {
    "bySelfDraw" : false,
    "label" : "Hong Kong · 8 faan",
    "score" : 96,
    "turns" : 14,
    "won" : true
  },
  "spec" : "1.1.0",
  "uses" : [
    "result"
  ]
}
```

Listing 4 — A deal carrying the challenger's result as a score-to-beat. “result” is advisory here (in uses, not requires), so a reader that ignores it still plays the deal.

7.4 A played result

© 2026 JSCBIZ. Open specification — free to implement (see §License).

```
{
  "format" : "mjpf",
  "game" : {
    "branchPointIndex" : 0,
    "humanSeat" : 0,
    "mode" : "play",
    "recordSeat" : 0,
    "ruleset" : "hongkong",
    "seats" : 4,
    "seed" : "12648430",
    "tier" : "veryGood"
  },
  "kind" : "result",
  "meta" : {
    "app" : "Mahjong Coach",
    "title" : "How I did"
  },
  "requires" : [
    "result"
  ],
  "result" : {
    "label" : "No win — exhausted",
    "score" : 0,
    "won" : false
  },
  "spec" : "1.1.0",
  "uses" : [

  ]
}
```

Listing 5 — A result report (kind:“result”, requires:[“result”]) sent back to a challenger.

8. Conformance guidance

8.1 The reader algorithm (normative)

- If `format` ≠ “mjpf” → refuse (*not MJPF*).
- Parse the MAJOR of `spec`. If MAJOR ≠ the reader's supported MAJOR → refuse (*incompatible major*). Differing MINOR/PATCH within the same MAJOR are accepted.
- If any entry of `requires` is not in the reader's known capability set → refuse (*unsupported capability*), reporting the unknown flags.
- Otherwise proceed. Ignore unknown `uses` entries, unknown top-level fields, and unknown `ext` keys.

8.2 Checklist for a conformant reader

- Base64-decode → **raw** inflate → JSON parse.
- Check `format == “mjpf”` and `spec` MAJOR == 1.
- Refuse if any `requires` flag is unknown (name it).
- Parse `game.seed` as a 64-bit integer from a string.

© 2026 JSCBIZ. Open specification — free to implement (see §License).

- Ignore unknown top-level fields, advisory `uses`, and unknown `ext` keys.
- Never read personal data (there is none) and never execute `ext` values.

8.3 Compatibility matrix

Document → / Reader ↓	doc 1.0.x	doc 1.<newer minor>	doc 2.x
reader 1.0.x	full	accepted; unknown fields & advisory caps ignored; refused only on an unknown required capability	refused (incompatibleMajor)
reader 1.<newer minor>	full	full	refused
reader 2.x	full (if 2.x keeps 1.x readable)	full	full

Table 7 — How a reader at one spec version handles a document at another.

8.4 Error vocabulary (reference)

```
notMJPF · malformedVersion · incompatibleMajor(found, supported)
unsupportedCapability([flags]) · unknownRuleSet(id) · badBase64
decompressFailed · badJSON
```

8.5 Security & privacy

MJPF carries **game data only** — seed, rule set, tier, mode, seats, and (for resume) the move log. It MUST NOT carry account identifiers, contact info, location, device ids, or any personal data. `meta.app`/`appVersion` are optional and non-identifying. A reader treats an imported document as untrusted input: validate the envelope, bound the event-log length, and never execute anything from `ext`.

9. Governance — the additive-only model

MJPF is designed to be a small, stable, boring format that outlives any one application. Its evolution is governed by a strict semantic-versioning policy whose central promise is **additive-only growth**.

Change class	What it means	Effect on old readers
PATCH	Editorial / clarification only; no change to the wire format.	None.
MINOR (additive)	New optional fields and new capability flags only. Nothing existing is removed or redefined.	Keep working — they ignore what they don't recognize and refuse only a document that requires a capability they lack.
MAJOR	A breaking change to the envelope or to the meaning of an existing field. Reserved for a last resort.	Refuse a higher major (incompatibleMajor).

Table 8 — The versioning policy. Capability flags are the primary evolution mechanism, so nearly all growth happens in MINOR versions without disturbing anyone.

© 2026 JSCBIZ. Open specification — free to implement (see §License).

The standing promise

A published MJPF specification version is **never changed retroactively**. Once a version is published, its wire format and the meaning of its existing fields are fixed for all time. Improvements are made by **adding** — new optional fields and new capability flags in a new MINOR version — so that every document ever produced against a published version remains valid and readable. We modify only going forward, with additive specifications; we never change what is already out there. Breaking that promise would require a MAJOR version bump, which readers explicitly refuse rather than silently misinterpret.

MJPF is currently **steward-maintained**. The steward's job is conservative: keep the current major stable, accept additive changes carefully, and never break old readers without a major bump. JSCBIZ, as the format's originator, coordinates the specification and its published versions. Proposals for additive capabilities are welcome via the project's public channels (see §“Contact & hosting”). A formal conformance / certification program (a hosted validator, a reusable CI action, an adapter contract, and a frozen test-vector corpus per version) is planned but parked, and documented separately for transparency.

10. Attribution

The Mahjong Portable Format (MJPF) and its transport encoding (MJQR) were **proposed, originated, and first specified by JSCBIZ**. JSCBIZ publishes the format openly and free of royalty so that the whole Mahjong software community may implement and interoperate with it; that openness does not diminish JSCBIZ's standing as the format's originator and first proposer.

Implementers and downstream specifications are asked (but not required) to acknowledge JSCBIZ as the originator of MJPF, for example: “Implements the Mahjong Portable Format (MJPF), originally proposed and specified by JSCBIZ.” Additive extensions and new capability flags contributed by others remain welcome and are credited to their authors, while the base format and its governance model remain attributed to JSCBIZ.

11. Licence

This section states the terms under which the Mahjong Portable Format is made available. It is written in plain open-specification language so that anyone may implement MJPF without restriction.

11.1 Grant of rights (royalty-free, open implementation)

JSCBIZ grants to everyone a worldwide, perpetual, irrevocable, non-exclusive, royalty-free, no-charge licence to:

- implement readers and writers of MJPF and MJQR in any product, commercial or non-commercial;

© 2026 JSCBIZ. Open specification — free to implement (see §License).

- use, copy, modify, and redistribute the specification text, the JSON Schema, and the example payloads, in whole or in part;
- create and publish independent implementations, adaptations, translations, and additive extensions of the format.

No fee, royalty, or separate permission is required to exercise these rights. This grant covers the format, its schema, and its example payloads.

11.2 Patents

JSCBIZ makes no patent claims over the Mahjong Portable Format. To the extent JSCBIZ holds any patent rights necessarily infringed by a conforming implementation of this specification, a worldwide, royalty-free, non-exclusive licence to those rights is granted for the purpose of implementing the format.

11.3 Names & marks

“MJPF” and “MJQR” name the format and its transport. Please apply these names only to implementations that conform to a published MJPF specification, and not to incompatible or forked formats, so that the names remain a reliable signal of interoperability. This is the only restriction on use of the names; it does not restrict implementation of the format itself.

11.4 Additive-only commitment

As set out in §“Governance”, published specification versions are not changed retroactively; the format evolves only by additive minor versions or, as a last resort, a major version that readers explicitly refuse. This commitment is part of the terms on which the format is offered: an implementation of a published version will not be invalidated by later revisions of that version.

11.5 No warranty

The specification is provided “AS IS”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event shall JSCBIZ be liable for any claim, damages, or other liability arising from, out of, or in connection with the specification or its use.

A permissive open licence (for example CC0 for the specification text and JSON Schema, and MIT for reference code) is compatible with the grant above; the final licence identifier is confirmed by JSCBIZ at publication time.

12. Contact & hosting

The specification, JSON Schema, example payloads, reference documentation, and this white paper are published at the canonical hosting location below. The site is a set of static files with no build step, so it can be hosted on object storage, any static host, or a documentation platform.

Resource	Location
Canonical website / downloads	`https://mjpf.openmahj.org`
Specification & JSON Schema	`https://mjpf.openmahj.org (spec + downloads)`

© 2026 JSCBIZ. Open specification — free to implement (see §License).

Resource	Location
Contact / proposals	`{{MJPF_CONTACT}}`
Originator	`JSCBIZ`

Table 9 — Where to find MJPF and how to reach the maintainers. The bracketed {{…}} token(s) are placeholders to be filled in at publication.

Publication note. `{{MJPF_CONTACT}}` is a placeholder — replace it with the final value before publishing this document.

— End of white paper. Mahjong Portable Format specification 1.2.0, proposed and originated by JSCBIZ. © 2026 JSCBIZ; open specification, free to implement.