Transaction types 24, 25, 26, 27, 28 and 39. Bodies, the app registry, payouts, and how to verify an outcome yourself. Transcribed from the node sources.

ZooBC On-Chain Apps

An **app** on ZooBC is a game or wager whose rules run in consensus. Every move is a transaction, the

board lives in chain state, and the payout is made by the protocol. There is no server to trust, no

operator who can refuse to pay, and no result anyone has to take on faith — the whole thing is

replayable from chain history by anybody, forever.

This manual is the integrator's reference: the six transaction types, the exact bodies, the app

registry with each app's move encoding, how money moves, and how randomness is derived and checked.

Everything here is transcribed from the node sources — `include/zoobc/common/types.h`,

`src/transaction/app_rules.cpp`, and `src/transaction/transaction_executor.cpp` — not from the design

notes, which predate the build and differ from it in places.

---

1. The three categories

Category	Seats	Counterparty	Randomness	Types
Solo vs the house	1	the protocol's apps pool	yes — block seed	16–21
Head-to-head	2	another player	only if the app uses it	1–8
Party	3–4	other players	dice from the block seed	32–35

All three share one engine: the same transaction types, the same stake escrow, the same per-move

deadline, and the same rule that one move is one transaction.

`seats` decides the category and is checked against the app type. A solo app must have `seats == 1`

and a type in 16–31; a multiplayer app must have `seats` 2–4 and a type outside that range. Getting

this wrong is rejected at mempool admission, not at execution.

---

2. Transaction types

Six types. All bodies are little-endian; all amounts are atomic (divide by 1e8 for ZBC).

Type	Name	Body
24	`CreateApp`	`app_type(1) · stake_token_id(8) · stake_amount(8) · seats(1) · params_len(2) · params · [opponent(36)] · [channel(1)]`
25	`JoinApp`	`app_id(8)`
26	`AppMove`	`app_id(8) · move_len(2) · move_bytes`
27	`ResignApp`	`app_id(8)`
28	`ClaimAppTimeout`	`app_id(8)`
39	`SettleApp`	`app_id(8) · final_seq(4) · move_count(4) · [seat(1) · move_len(2) · move · signature(64)]*`

The two optional trailing fields on CreateApp

`opponent` and `channel` are both optional, and which are present is worked out from how many bytes

remain after `params`:

Bytes remaining	Meaning
0	open app, on-chain play

Bytes remaining	Meaning
1	`channel` only
36	`opponent` only
37	`opponent` then `channel`

An `opponent` turns the app into a direct challenge — only that address may join. Omit it and anyone

may take the seat. `channel = 1` marks a state-channel app and is valid only when `seats == 2`.

> Multiplayer apps store players in fixed 36-byte slots, so every seated address must be 36-byte canonical (a ZBC address or a bare 32-byte public key). Non-36-byte accounts are rejected at create. Solo apps have no such restriction — there is only one player.

---

3. The app registry

`app_type` is a single byte. Only these values are known; anything else is rejected.

Head-to-head (seats 2–4)

Type	App	Board / state	Move bytes
1	Tic-tac-toe	9 cells	`[cell 0..8]`
2	Chess	64 squares	`[from, to]` — full legality, check, checkmate, stalemate, auto-queen promotion
3	Connect-4	42 cells (7×6)	`[column 0..6]` — gravity, 4 in a row
4	Checkers	64 cells + multi-jump lock	`[from, to]` — forced capture, multi-jump continuation, kings
5	Reversi	64 cells	`[cell 0..63]` — must flip at least one; a side with no move passes; most pieces wins

Type	App	Board / state	Move bytes
6	Gomoku	225 cells (15×15)	`[x, y]` — 5 in a row
7	Battleship	6400 commitment + 200 reveal	commit-reveal, below
8	Dots-and-boxes	24 edges + 9 boxes	`[edge 0..23]` — completing a box moves again

Solo vs the house

Type	App	`params`	Pays
16	Two dice	`[bet_type 0..3, total]`	under 7 / over 7 **2.28×**, lucky 7 **5.7×**, exact total **3420/ways %**
17	Coin flip	`[choice 0/1]`	**1.98×**
18	Roulette	`[bet_type, value]`	single number **36×**, colour **2×**
19	Slots	none	three of a kind **15×**, triple 7 **50×**, any pair **1.8×**
20	Lottery	`[pick 0..99]`	**90×**
21	Crash	`[target ×100, 2 bytes LE]`	**target ×**, 1.01× to 10.00×

Bet types for dice are `0` under 7, `1` lucky 7, `2` over 7, `3` exact total. For an exact total the

multiplier is `3420 / ways` in hundredths, where `ways = 6 - |7 - total|` — so 7 pays 5.70× and 2 or

12 pay 34.20×. Roulette has 37 pockets; `0` is green and loses both colours.

Party (seats 3–4)

Type	App	State	Move bytes
32	Ludo	`seats×4` token positions + pending die	two-phase: roll, then choose a token

Type	App	State	Move bytes
33	Pig	`seats` scores + turn total	`[0 roll, 1 hold]`
34	Race (snakes & ladders)	`seats` positions	`[0]` — roll
35	Monopoly-lite	money, positions, ownership, bankruptcy, phase	two-phase: roll, then optionally buy

---

4. Money

Stakes. `CreateApp` escrows the creator's stake. Each `JoinApp` escrows an equal stake. The pot

is the sum. `stake_token_id` of `0` means ZBC; any other value stakes that colored token, and the

whole app — pot, rake, payout — settles in that token.

Rake. On a decisive end, **1% of the pot** goes to the apps pool and the winner receives the

rest. On a draw or a cancelled app, every stake is refunded and no rake is taken.

The apps pool is a protocol account with one balance per token. It grows from that 1% rake plus

the house edge on solo apps. It is the counterparty for every solo bet, so its balance is what makes

solo play possible at all — you can read it at any time:

```
GET /api/v1/apps/pool
```

Bankroll cap. A single solo bet's maximum possible payout may not exceed **5% of the pool** for

that token. This is checked against the bet's own worst case for the house, not its average, so no

single hit can drain the pool. A bet that would breach it is rejected — if a large bet is refused,

this is usually why.

---

5. Randomness, and how to check it

Solo apps resolve from the block seed, which on ZooBC is

`block_seed = blocksmith_signature(SHA3(previous_block.block_seed))` — deterministic, publicly

verifiable against the blocksmith's public key, and impossible to know before the block exists.

A bet placed in the block at height `H` resolves at `H + 2`:

```
r = SHA3-256( block_seed[H+2] ‖ app_id )   →   the low 8 bytes, little-endian, as a uin
```

The player commits two blocks before the seed they are resolved against exists, so they cannot

predict it or pick it. Anyone can recompute `r` afterwards from the block seed and the app id, and

therefore recompute the outcome. Nothing about the result depends on data the chain does not hold.

Crash derives its crash point from the same `r`:

```
u = r mod 1e6
C = 99_000_000 / (1_000_000 - u)      clamped to [100, 1000], i.e. 1.00× .. 10.00×
```

The bet wins `stake × target / 100` when `C >= target`. The `99` in the numerator is the house

edge — 1%. The 10.00× clamp is the ceiling, so the largest possible crash win is ten times the bet.

Multi-play. Any solo app may be played up to **100 times from one bet**. Append a trailing count

byte `N` to `params`, after that app's choice bytes. The stake is split evenly across the `N` plays

and the payouts summed, so one transaction and one reveal produce `N` independent outcomes without

waiting a block each. Omitting `N` means a single play, and a single play is bit-identical to what it

would have been before multi-play existed. `N` must be 1–100 and the stake must divide so each play

stakes at least one unit.

---

6. Turns, deadlines, and endings

An app becomes active when its last seat is filled. From then on each accepted move sets

```
deadline_height = current_height + 240        (~1 hour at 15 s per block)
```

A move is accepted only from the player whose turn it is, only while the app is active, and only

before the deadline passes. There are four ways an app ends:

- **A win or a draw in the rules.** Settled immediately — winner paid, or all stakes refunded on a draw.
- `ResignApp`**.** You forfeit; your share of the pot goes to the opponent.
- `ClaimAppTimeout`**.** Your opponent let the 240-block deadline pass. You claim, and you win the pot.

The claim is yours to make — nothing happens automatically just because the deadline elapsed.

- **Nobody joins.** An open app that is never taken up is cancelled and the stake refunded.

7. State channels and SettleApp

For a head-to-head app, playing every move on-chain costs a transaction and a block per move. A

state channel trades that for a single settlement: create with `channel = 1` and `seats = 2`, play

off-chain with each side signing every move, and submit the whole game once as `SettleApp` (type 39).

The chain replays the ordered signed moves from the app's initial state through the same rules

engine, verifies each signature and each turn, and pays the winner. A settlement can be overridden by

one carrying a higher `final_seq` for **240 blocks** after it lands; once that challenge window

passes, it settles.

Three limits are enforced on the body, and an integrator has to respect all three:

- `final_seq` must **equal** `move_count`.
- `move_count` may not exceed **1024** entries.
- `move_count` may not exceed what the remaining body can physically hold. Each entry is at least 67

bytes — `seat(1) + move_len(2) + signature(64)` — so a count larger than `remaining / 67` is

rejected before a single byte is allocated.

> 1024 entries is a per-ply limit, not a per-move-pair one: one entry is one ply, so a 1024-entry settlement covers a 512-move game. The longest chess game ever recorded ran 269 moves — 538 plies — so the cap is roughly double the longest game anyone has actually played.

8. Battleship (type 7)

Battleship is the one head-to-head app that needs hidden information, so it uses commit-reveal.

The board is 10×10. For each of the 100 cells the player picks a random 32-byte salt and commits

`SHA3(is_ship ‖ salt)`; the commitment is those 100 hashes concatenated — **3200 bytes**, passed as

`params` on create. `seats` must be 2 and the commitment must be exactly 3200 bytes or the create is

rejected.

Play alternates two move forms:

- **Fire** `[0, cell]` — you fire at a cell on the opponent's board. The turn passes to them to reveal.
- **Reveal** `[1, cell, is_ship, salt(32)]` — they prove what was there by opening that one cell's

commitment. Because each cell is committed separately, a player cannot lie about a single cell.

Hit or miss is recorded, and the revealer fires next.

Hit all 17 ship cells and you win. A player who will not reveal is on the clock like any other move,

so `ClaimAppTimeout` applies.

> **Fees scale with size.** A 3200-byte create is a large transaction and the fee floor scales with transaction size. On a live chain a 0.1 ZBC fee was rejected as too low and roughly 5 ZBC was accepted. Budget for it on the create; fire and reveal moves are small and cheap.

---

9. Reading app state

Four endpoints, served by the **node API** — note that these are node endpoints, not archival ones:

Endpoint	Returns		
`GET /api/v1/apps`	the lobby; filter with `status=`, `category=solo\`	pvp\	party`, limit=`
`GET /api/v1/apps/open`	open head-to-head challenges anyone may join		
`GET /api/v1/apps/:id`	one app's full state		
`GET /api/v1/apps/pool`	the apps pool balances, per token		
`GET /api/v1/apps/stats`	live counts: total, open, active, finished, players, pot at stake		

An app row carries `id`, `app_type`, `status` (0 open, 1 active, 2 finished, 3 cancelled), `seats`,

`creator_address`, `players`, `opponent_address`, `stake_token_id`, `stake_amount`, `pot`,

`state_blob`, `turn`, `created_height`, `last_move_height`, `deadline_height`, `resolve_height`,

`winner_address`, `persist_height` and `channel`.

The `state_blob` is the current derived board, kept so a node need not replay history every block.

It is a convenience, not the record: the record is the move transactions, and a client can replay

them for audit or animation. A finished app's live row is pruned after a grace period; its moves stay

in chain history permanently, so the app is always reconstructable.

---

10. From the command line

`zoobc-cli` covers all six types. The first parameter is always the sender's private key.

Command	Type
`app-create`	24

Command	Type
`app-join`	25
`app-move`	26
`app-resign`	27
`app-claim`	28
`app-settle`	39

```
# a coin-flip against the house: type 17, 1 ZBC, seats=1, choice 0
zoobc-cli app-create <privkey> 17 0 100000000 1 00 --api $API

# tic-tac-toe, open to anyone, 1 ZBC
zoobc-cli app-create <privkey> 1 0 100000000 2 --api $API

# take the middle square
zoobc-cli app-move <privkey> <app_id> 04 --api $API
```

`zoobc-cli help app-create` lists the fields for any of them. See the **CLI Manual** for the full

command set and the **Transaction Manual** for byte layouts across all 50 transaction types.

---

11. Constants

Constant	Value	What it governs
Rake	1% of the pot	to the apps pool, on a decisive end only
Move deadline	240 blocks	~1 hour; reset by every accepted move
Solo resolution	bet height + 2	~30 seconds
Bankroll cap	5% of the pool	maximum payout of any single solo bet
Multi-play cap	100	plays per solo bet

Constant	Value	What it governs
Settle moves cap	1024	entries per `SettleApp`, ≥67 bytes each
Settle challenge window	240 blocks	~1 hour during which a higher `final_seq` wins
Seats	1, or 2–4	1 is solo; 2–4 is multiplayer
Battleship commitment	3200 bytes	exactly, or the create is rejected