

WEATHER MARKET **V3** MIGRATED UPDATED 2026-09

Weather Market protocol whitepaper

Technical Whitepaper · v3 (2026-09)

A consensus weather oracle that sells determinations, not weather data: whether a condition held at an hour, decided by a fixed rule over eleven sources, anchored on Flare every hour and published there as a verdict any contract can read, paid for in XRP on the XRP Ledger, and purchasable by people and autonomous agents without a human in the loop.

Abstract

Weather is one of the few real-world quantities that contracts settle on directly: crop insurance, energy hedges, event cancellation cover, shipping and construction delay clauses. Yet the data those contracts reference is almost always a single provider's number. If that provider is wrong, offline, or adversarial, the settlement is wrong and there is no way to prove it after the fact.

Weather Data Market treats weather the way a price oracle treats an asset price. Every hour it collects the same measurement from eleven meteorological source feeds, rejects outliers by a fixed rule rather than by trust, forms a consensus value, checks it against independent observation, and writes the hash of the full run to Flare mainnet in the SnapshotAnchor contract - one transaction per run, never overwritten, so the first block timestamp is the record. Until 23 September 2026 a first-generation copy of the same hash was also written on BNB Chain; those anchors remain verifiable. What it sells is the determination that follows: whether a stated condition held at that hour - met, not met, or withheld - with the reasons, published on Flare in a registry that marks a verdict settlement-grade only when its backing run is anchored. The rule's parameters are proprietary; what any counterparty can check is that the rule did not change and that the run existed, hashed, before the dispute did.

1. The problem

A settlement oracle has to answer one question convincingly: why should the losing side of a contract accept this number? Single-source weather feeds fail that test in three specific ways.

Unverifiability. An API returns a value. Nothing about that response proves it was the value served at the time of settlement, or that it was not edited afterwards.

Single point of failure. Meteorological models genuinely disagree - a station outage, an ingest bug, or a bad grid interpolation can move one provider by several degrees while the others are unchanged. A contract referencing that one provider settles wrong.

Human-shaped access. Weather APIs are sold through sales calls, monthly invoices and dashboards. An autonomous agent that wants tomorrow's rainfall determination for one city cannot buy it at 3am without a human signing something.

2. System overview

```

11 source feeds (3 direct, 8 via Open-Meteo)
  | hourly collection (staged to respect rate limits)
  ▼
median + MAD outlier rejection          ← statistical, not reputational
  |
  ├── consensus index (T, H, W, P) + WDI / TDI
  ├── corroboration verdict vs the observation class
  |
  ▼
canonical JSON → SHA-256 → anchored every run
  |                                     ├── Flare: SnapshotAnchor.anchor(sha256, date, runId)
  |                                     |   anchor of record - block timestamp, never
  |
overwritten
  |                                     ├── BNB Chain: IndexAnchored event (first generation)
  |                                     └── (legacy, runs before 2026-09-23: XRP Ledger memo,
  |                                         provable on Flare via FDC XRPPayment)
  |
  ▼
verdicts → Flare: DeterminationRegistry (met / not met / withheld)
  |       └── settlement-grade when SnapshotAnchor.isAnchored(sha256) is true
  |
  ▼
settlement layer: XRP (XRP Ledger, priced by FTS0v2 XRP/USD) - USDT on BNB retired 2026-
09-23
  └── humans via wallet · agents via REST + prepaid 402 metering
  
```

3. Data acquisition

Eleven sources are collected for 167 cities every hour. They are deliberately heterogeneous: national meteorological services that run their own physics, direct observation networks, and

one vendor blend. Correlated sources would defeat the purpose - outlier rejection only works if the sources can disagree independently. Heterogeneous in origin is not the same as independent in delivery: three feeds are fetched directly (KWeather, METAR, MET Norway) while the seven agency models and the vendor blend all arrive through Open-Meteo, so an outage there removes eight at once even though the underlying physics remain distinct.

Underneath this sits a principle worth stating outright: a city's representative value is not one station's reading. Stations differ by microclimate, siting and elevation, and an airport several kilometres out is not the city centre. A defensible city index therefore combines many observation streams - the city's official stations, ICAO METAR, GTS bulletins - with numerical models from several national services, and, where the provider supports it, AI methods and terrain information such as land cover. KWeather builds its city indices this way, which is why its readings enter here as an observation-class source rather than as one more model. The corollary is that this is not a fixed ceiling: as observation networks densify, the same method yields a more accurate index.

Forecasts follow the same principle. Relying on eleven sources for observations while taking forecasts from a single vendor would leave the weaker half of the product resting on exactly the assumption the rest is built to avoid, so forecasts are merged from two independent providers - Open-Meteo and KWeather. Overlapping days are averaged and every row carries the number of contributing sources, so a single-source day is never mistaken for a corroborated one.

Source	Kind	Notes
--------	------	-------

---	---	---
-----	-----	-----

METAR (NOAA AWC)	observation	Airport surface observations - measured, not modelled. ICAO stations are mapped to every city in the catalogue. One of the two members of the observation class the consensus is checked against - the other, KWeather's SYNOP feed, draws on a different station network - never the sole definition of the city.
------------------	-------------	--

NOAA GFS	model	United States
----------	-------	---------------

DWD ICON	model	Germany
----------	-------	---------

JMA	model	Japan
-----	-------	-------

Météo-France ARPEGE/AROME	model	France
---------------------------	-------	--------

| ECMWF IFS | **model** | Europe (intergovernmental) |

| UK Met Office | **model** | United Kingdom |

| ECCC GEM | **model** | Canada |

| MET Norway | **observation** | Norwegian Meteorological Institute - global numerical model (locationforecast). Its own observing network covers Norway and the Nordics, not the other cities here. |

| Open-Meteo best_match | **synthetic** | Vendor's own blend - treated as one opinion, not as truth |

| KWeather | **observation** | Korea. City indices built from official stations, ICAO METAR and GTS bulletins combined with ECMWF/GFS/JMA/KIM models, AI methods and terrain data; also runs its own particulate sensor network and supplies a second forecast source. Outside Korea, KWeather re-provides GTS SYNOP bulletins measured by WMO synoptic stations in those countries - the KMA observes South Korea only |

3.1 Rate limits as a design constraint

Fetching eleven sources for 167 cities every hour exceeds the per-minute quota of the upstream aggregator, and the naive fix - rotating which models participate - silently degrades the consensus, because a rotated-out source contributes nothing to that hour's median.

Instead the system pre-collects. A five-minute cron refreshes the two stalest national models into a staging table, so all seven complete a rotation roughly every eighteen minutes. The hourly consensus run then merges the staged observations - accepting anything fresher than 75 minutes - with the live sources. Every source participates in every run, and no minute ever carries more than two batches. Selecting by staleness rather than by a fixed time slot matters: serverless crons are jittery, and a fixed slot means one missed invocation drops that model for a full hour.

4. Consensus

Each metric is reduced independently. The estimator is the median with outlier rejection by median absolute deviation - the standard robust construction, chosen because it has a

bounded breakdown point: up to half the sources can be arbitrarily wrong without moving the result.

```
// per-metric consensus across the collected source feeds
center    = robust center of the values           // not a mean - one feed cannot drag it
spread    = robust dispersion around that center
threshold = f(spread, floor_metric)              // parameters: proprietary, fixed per
version
outliers  = { v : |v - center| > threshold }      // rejected, recorded by name
index     = robust center of the remaining values

// floor_metric keeps unanimous rounding differences from reading as outliers.
// Every run's output is hashed and anchored on-chain; the rule does not change
// between runs without a versioned notice.
```

Rejected sources are not silently dropped. Each run records which source deviated, by how much, and in which direction, and subscribers see it on the dashboard and in the per-source feed. A source that is routinely rejected is visible to every paying counterparty, which is the accountability a single-provider feed cannot offer. The rule that decides rejection is fixed per version and its parameters are not published.

4.1 Corroboration verdict

Consensus among models is not the same as agreement with reality, so every run is additionally checked against measurement. What it is checked against matters: a single airport station is not the city. METAR sits kilometres from the centre in a different microclimate, and treating that one point as ground truth measures the station rather than the city. The check therefore runs against the whole observation class - airport METAR and KWeather, which re-provides official city stations and GTS bulletins - each compared with the consensus inside fixed tolerances, which are part of the proprietary rule and are not published.

A run is verified when more fresh observations agree with the consensus than disagree and at least two agree; single_source when one agreed and none dissented, which is real corroboration but thin; mixed when agreement and dissent are tied, which is undecided rather than a pass; mismatch when measurement is available but mostly diverges; and unverified when no fresh observation-class source could be read. The counts of agreeing, disagreeing and abstaining observations ship with the verdict, so a settlement can demand not just a favourable label but a stated depth of corroboration. Because the basis is a class rather than a fixed

station, adding observation networks makes the verdict stronger without changing the rule - accuracy grows with station density.

5. Indices

Two indices are published alongside the raw metrics. They exist because settlement contracts want a single scalar with a fixed definition, not a judgement call.

```
// deterministic indices over the consensus values
WDI = weighted composite of normalised T, H, W, P // weights and ranges: proprietary,
fixed per version (wdi_v1)
TDI = normalised temperature // fixed range, fixed per version
(tdi_v1)

// Same inputs, same number, every time. A change to any coefficient or range
// ships as a new version; contracts written against a version stay computable.
```

WDI (Weather Data Index) is a composite; TDI (Temperature Data Index) isolates the temperature component for contracts that only care about heat. Both are pure functions of the consensus values with fixed weights and bounds that do not change per city or per season. The coefficients are part of the proprietary rule; each index carries a version, and a change to any coefficient ships as a new version so contracts written against the old one stay computable.

6. Verifiability

Three independent layers make the output checkable. Each answers a different question.

6.1 Canonical serialization and anchoring

Every run is serialized to a canonical JSON byte sequence and hashed. The download endpoint returns exactly those bytes, so the hash a buyer computes locally is the hash the server published. The anchor of record is on Flare mainnet: the publisher key calls `SnapshotAnchor.anchor(sha256, date, runId)` once per run, the contract stores the block number and timestamp, and it refuses to overwrite a hash that already exists - the first timestamp is the evidence. `DeterminationRegistry` reads `isAnchored(sha256)` from the same contract, so a verdict becomes settlement-grade the moment its run is anchored. Until 23 September 2026 a

first-generation copy was also written on BNB Chain as an IndexAnchored event; those 1,452 anchors remain verifiable. Runs before 23 September 2026 also carry a legacy anchor in the memo of a one-drop XRP Ledger payment, provable on Flare through XrplAnchorVerifier and an FDC XRPPayment attestation; that route was retired for new runs once anchoring directly on Flare cost one transaction instead of one attestation. What any anchor proves is the same: this hash was published by our key at a public timestamp, before any dispute existed.

```
// Reproducing a purchased dataset byte-for-byte
1. GET /api/index/snapshot/{runId}      → canonical JSON (exact bytes hashed)
2. sha256(bytes)                        → compare with the value you were given
3. SnapshotAnchor.isAnchored(hash)      → Flare mainnet - the anchor of record;
   SnapshotAnchor.anchors(hash)         block number and timestamp it was fixed at
4. WeatherOracleAnchor (IndexAnchored) → the same hash, first-generation, on BNB Chain
5. Runs before 2026-09-23 only: the same hash in the memo of a 1-drop XRP Ledger
   payment, provable on Flare via XrplAnchorVerifier and an FDC XRPPayment attestation
6. The anchoring block predates any dispute, so the data could not have been
   altered after the fact. DeterminationRegistry reads step 3 to mark a verdict
   settlement-grade - a contract needs nothing from us to check it.
```

6.2 FCC - attesting the computation

Anchoring proves that a run existed and was not altered afterwards, but not that we ran the rule honestly in the first place. Flare's Confidential Compute runs the consensus inside a trusted execution environment; the enclave signs its output, and WeatherFccConsumer verifies that signature chain on chain before storing anything. The enclave's code hash is what a counterparty checks - the rule inside it stays confidential.

```
resultHash = keccak256(abi.encodePacked(
    keccak256(resultData), actionId, keccak256(bytes(tag)), status));
digest     = keccak256(abi.encode("TEE_ACTION_RESULT", block.chainid, resultHash));
require(ecrecover(EIP191(digest), v, r, s) == teeAddress);
```

Verified end to end on Coston2: a correctly signed result is accepted and stored, a forged signature reverts with InvalidTeeSignature, and a replay reverts with AlreadyRelayed.

Confidential settlement: transparent, not visible

The attested consensus above still publishes the value it attests. For settlement, publishing is the leak: a threshold and an index on-chain expose a contract's position to its counterparty.

The blind path keeps the proof and removes the numbers. The requester commits to a condition and seals it to the enclave; the enclave evaluates it against its own eleven-source consensus and signs an outcome bit, the condition commitment, an index commitment and a report sealed to the requester. Fewer than three sources yields 'no verdict', never 'not met'. Any party holding the salt can open a commitment in a dispute - the design makes disclosure possible, not automatic.

```
requester  seal(condition) -> enclave pubkey      commitment = keccak(cityId, metric, op,
thresholdE2, salt)
enclave    open · collect 11 sources · median+MAD · evaluate
           sign(cityId, evaluatedAt, commitment, outcome, settleable, nSources,
               indexCommitment, sealedReport -> requester)
chain      stores bits + two commitments. No threshold, no index.
```

7. Settlement

Purchases settle as contract calls rather than through an invoicing workflow. The transaction hash is the payment record and the technical contract identifier; tax invoices, receipts and other statutory documents are issued separately on request under the applicable law. A quote binds product, cities and date range to a quoted; the buyer settles on chain; the server verifies the emitted event against the quote before releasing data. Multi-city purchases settle in a single transaction, with citiesHash binding the exact city set to the payment.

Pricing is by period rather than a flat rate: 25 for 30 days or 200 for a year, per city, in USD - paid on the XRP Ledger in native XRP, the USD price converted at the FTSOv2 XRP/USD rate on Flare when the quote is issued and fixed for the quote's ten minutes. A subscription runs for one of those two terms only. It opens everything accumulated for that city without limit - history, per-source readings, five-minute observations, the full forecast, the verdict on every hour - and includes determinations on it. Metered credits at 0.04 USD per call open the same data for any city, one credit per call, which is the form for republishing to one's own customers at volume: the bill follows the calls.

On Flare the same catalogue is payable in native FLR, with the USD price converted at the FTSOv2 on-chain feed at execution time and any overpayment refunded in the same transaction. Nothing off-chain decides the exchange rate. The XRP Ledger has no contract to recompute the price, so the binding is made differently: a DestinationTag issued with the quote

ties the payment to that order, and the currency, the issuer and the amount actually delivered - not the amount sent - are all verified against the ledger itself.

7.1 Forward contracts

Every purchase is forward-only: a range starts today or later and runs up to a year. We do not sell the past. Future days are delivered as null rows that fill in as the data accumulates, against a stable download URL. This is what makes the product usable as a settlement reference for a contract written today and settled next season.

7.2 Access model and the determination product

The free tier is the present: the current consensus value for any named city and the next three days of forecast, up to 120 requests an hour. Everything accumulated is a paid product - the consensus history, every source's raw reading, the five-minute observations, the canonical snapshot bytes, the full sixteen-day forecast and the verdict itself. The boundary is between now and the archive, not between value and verdict: an archive of free responses would let anyone reconstruct the determination and republish it, and that is the product.

A determination is the unit we sell. The buyer opens a condition - a city, a metric, a comparison, a threshold and a target hour in UTC - and after that hour the next run answers one of three ways. Met: the condition held and the consensus was corroborated well enough to settle on. Not met: it did not hold, under the same standard. Withheld: no verdict, because independent observations were split, too few sources contributed, or no run exists for that hour. Withheld is a result rather than a failure; it says the hour cannot bear a settlement, which is precisely the information a losing side needs to accept before it accepts a number.

Every outcome carries a report: the consensus value at that hour, the condition, how many independent observations supported the consensus and how many disagreed, how many sources contributed and how many were excluded, and the reason in plain language. The report explains the outcome without disclosing the rule. A determination is included with a subscription covering the city and otherwise paid per determination from metered credits; a request without a credential answers 402 with the price and the top-up flow, which is how an autonomous agent opens and pays for one on its own. Determinations are final: the same condition and hour read the same anchored run and return the same outcome.

8. Machine-native commerce

Agents are first-class buyers, not a special case. There is no signup, no API key issuance flow, and no human approval anywhere in the purchase path - the wallet is the identity and the payment is the authorization.

```
// Autonomous purchase - the entire flow
quote = GET /api/quote?cities=seoul,tokyo&from=...&to=...
        → { quoteIdHash, cityIds, amountWei, contract, usdt, chainId }
approve(usdt, contract, amountWei)
purchaseBundle(quoteIdHash, cityIds, fromDate, toDate) // one transaction
claim = POST /api/purchases/claim { txHash, quoteId }
        → dataset + sha256 + permanent downloadUrl

// Or pay in XRP: the same quote carries an "xrpl" block
quote.xrpl → { destination, destinationTag, amount: { drops }, rate: { usdPerXrp, source } }
send ≥ amount.drops of XRP with that destinationTag // any XRPL wallet, no approve
step
claim = POST /api/purchases/claim { txHash, quoteId, chain: "xrpl" }
```

For per-call use the same account sends XRP to the payment address with DestinationTag 402 and posts the transaction hash, signed by that account; the server verifies the payment against the validated ledger, converts the delivered XRP at the FTSOv2 rate of that moment, and issues call credits. When credits run out the metered endpoint answers 402 Payment Required with machine-readable payment instructions, so an agent can top itself up and continue without intervention.

The same tools are reachable over MCP at /api/mcp, so an agent running in Cursor, Claude Desktop or any Streamable HTTP host can read the current consensus, and - with the same credential it uses over HTTP - the per-source values behind it and the snapshot verification evidence, without writing HTTP calls. The current value needs no key; everything accumulated, and every determination, goes through the wallet flow above.

9. Security model

The server never handles customer keys - all signing happens in the buyer's wallet, and the backend only verifies transaction receipts or ledger entries. The one server-held key publishes snapshot anchors and verdicts and can do nothing else; it cannot move funds.

Payment integrity rests on the contract event, not on trust in the API: the claim endpoint re-derives the expected quote hash, city hash, date range and amount from the receipt and rejects any mismatch. Replay is prevented by a uniqueness constraint on the transaction hash. Collection endpoints are secret-gated because an unauthenticated caller could otherwise exhaust upstream quotas and stall the oracle - a denial-of-data attack that costs the attacker nothing.

10. Deployments

Settlement runs on two mainnets. Flare mainnet carries the anchor of record (SnapshotAnchor), the verdict registry (DeterminationRegistry) and, for runs before 23 September 2026, the legacy anchor verifier (XrplAnchorVerifier); the XRP Ledger takes XRP payments, priced from the FTSOv2 XRP/USD feed on Flare when a quote is issued. BNB Smart Chain, which carried USDT settlement and first-generation anchors until 23 September 2026, is retired: its 1,452 anchors remain verifiable and its purchases remain downloadable, but no new transactions are made there. Only the FTSOv2 market and the FCC consumer remain on Coston2, because FCC is still pre-production. Source is verified on the block explorers, and addresses are configuration rather than build constants, so moving a component between networks is a settings change rather than a redeploy.

| Component | Network | Address |

| --- | --- | --- |

| Legacy anchor (until 2026-09-23) | BNB Chain (56) | weathermarket.io/whitepaper |

| XRP payment | XRP Ledger (mainnet) | weathermarket.io/whitepaper |

| Legacy memo anchor (until 2026-09-23) | XRP Ledger (mainnet) | weathermarket.io/whitepaper |

| Snapshot anchor | Flare (14) | weathermarket.io/whitepaper |

| Anchor proof (FDC) | Flare (14) | weathermarket.io/whitepaper |

| Verdict registry | Flare (14) | weathermarket.io/whitepaper |

| FTSOv2 market | Flare (14) | weathermarket.io/whitepaper |

| FCC consumer | Flare (14) | weathermarket.io/whitepaper |

Addresses come from the live runtime configuration and are not duplicated here; the live whitepaper (linked above), section 10, carries the current values.

11. Status and limitations

Stating plainly what is proven and what is not, because a settlement oracle that overstates itself is worse than one that is merely incomplete.

Working and verified on chain: hourly consensus across all eleven sources; cross-check verdicts; canonical hashing anchored on Flare mainnet for every run, back-filled to the first run; XRP subscription settlement on XRPL mainnet, priced by FTSOv2 at quote time, bound to the quote by DestinationTag and checked against the drops actually delivered; the verdict registry live on Flare mainnet, settlement-grade for every verdict whose run is anchored; FLR settlement priced by the live FTSOv2 feed with refund; the FCC signature chain including rejection of forged signatures and replays; per-call metering with 402 payment instructions. Purchase, claim, download and metering have been exercised end to end by both a wallet user and an autonomous agent.

Not yet complete. The FCC path currently signs with a test key rather than a production enclave, so the computation proof is demonstrated but not yet rooted in real attested hardware; it and the FTSOv2 market stay on Coston2 until FCC reaches production on Flare. Blind settlement is written and not deployed. Payments are native XRP, which FDC can attest, but no contract yet consumes a payment proof: payments are verified against the ledger by the server. RLUSD is not accepted yet - FDC attests native XRP payments only, so an RLUSD payment could not be proven to a Flare contract until an issued-currency attestation exists. The anchor is written by our key, as every anchor is; its value is the public timestamp, not the author, and the parts that do not depend on us are the verdict registry anyone can read and the enclave code hash anyone can rebuild. The consensus is only as independent as its sources: seven of the model feeds arrive through one aggregator, so an outage there degrades diversity even though the reference observation and KWeather remain independent.

Appendix - interfaces

Full endpoint reference and runnable agent examples are in the [developer documentation](#).
Live contract registry, latest anchor evidence and reproduction steps are on the [technical proof](#)
page. The public consensus feed is open and unauthenticated at `/api/index`

**Weather Data Market · consensus weather oracle · sells determinations, not data · anchor
of record and verdict registry on Flare · XRP on the XRP Ledger · every run hash-anchored,
the rule fixed and proprietary**