ScirDom Algorithm Specification

Algorithm identifier	`scirdom.alg.v1.0`
Document version	`1.0`
Published	`2026-04-03`
Last updated	`2026-04-19`
Draw mode	`unweighted_without_replacement`
Winner model	Exact direct winner-selection sequence without replacement

This document defines the published method ScirDom uses to select winners in an unweighted draw without replacement. It is written for operators, auditors, regulators, and technical reviewers who want to inspect the method or reproduce a draw independently.

A ScirDom draw is fully determined by three things: the locked participant register, the disclosed starting offset within the active entropy block, and the rules in this document. Given those inputs, an independent reviewer should be able to reproduce the result without access to ScirDom systems.

Winners under ASD v1.0 are published in the exact order they are selected. The published winner list is therefore the certified selection sequence. It does not imply prize rank unless a separate draw rule explicitly assigns prize positions onto that sequence.

Files in the public ASD release bundle:

File	Description
`ScirDom_AlgSpec_v1_0.pdf`	Public specification document
`scirdom_asd_v1.py`	Python reference implementation
`scirdom_asd_v1.js`	JavaScript reference implementation
`scirdom_asd_v1_vectors.json`	Machine-readable test vectors
`ScirDom_How_To_Verify_A_Draw_v1_0.pdf`	Public first-principled replay guide

These five filenames define the published ASD package for `scirdom.alg.v1.0`. They are the public files a reviewer should expect to download from the ScirDom website when verifying the method independently.

The five-file bundle supports public conformance checking of the published reference implementations and the published synthetic test vectors. A specific certified draw is replayed from its draw evidence package, which supplies the locked participant register, the draw-specific entropy bytes, the recorded ASD identifier, and the recorded ASD document hash.

---

1. Document Status and Versioning

This document is the published definition of the winner-selection method identified as `scirdom.alg.v1.0`.

ASD v1.0 is the current published definition used in the Phase 7 public beta. Because ScirDom remains pre-release until final public launch, corrective updates may still be issued to this version while the beta remains open.

Once ScirDom is live, any behavioural change to the selection method will require a new ASD version with a new identifier, updated reference files, and new test vectors. Historical draws remain governed by the ASD version recorded in their evidence pack.

The evidence pack for each draw records the document filename and SHA-256 of the exact public ASD file used for that draw.

---

2. Normative Language and Conformance

The key words "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", and "MAY" are to be interpreted as normative requirements within this specification.

A **conforming implementation** is any implementation that, given the same raw participant entries, the same `offset_start`, the same entropy bytes, and the same K, produces outputs identical to this specification and to the published machine-readable vectors.

The written ASD and the machine-readable vector file are the source of truth. The Python and JavaScript implementations are reference implementations only.

A conforming implementation MUST preserve the following invariants:

1. Locked participant-list construction is deterministic and language-neutral.
2. Winner selection is exact and unbiased.
3. Winners are treated as an exact ordered selection sequence.
4. Any published winner list preserves the actual selection order exactly.
5. Rejected entropy bytes are still consumed.
6. No fixed byte-count formula may override or weaken the rejection-sampling semantics defined here.

---

3. Input Model

This section separates the literal ASD execution inputs from surrounding audited draw-record metadata. The public Python and JavaScript reference implementations accept only the execution inputs.

3.1 Execution inputs

Input	Type	Description
`participant_entries`	List of records	Parsed participant records supplied to the ASD in the form `{participant_id, name}`. CSV first-column parsing, recognised-header skipping, and manual line parsing are pre-ASD ingestion steps. `participant_id` is a draw-local sequential identifier such as `P-0001`. `name` MUST already be a valid UTF-8 string. Invalid UTF-8 is a pre-ASD ingestion error.
K	Positive integer	Number of winners to select. MUST be at least 1.
`offset_start`	Non-negative integer	Byte offset in the entropy source at which this draw begins consuming bytes.
`entropy_bytes`	Byte sequence	The entropy source made available to the ASD. For a specific certified draw replay this is the disclosed draw-specific entropy from the evidence pack. For the published public vectors this is the deterministic synthetic test entropy defined in Section 12.

This ASD defines the `unweighted_without_replacement` mode only. Weighted draws, with-replacement draws, or any other mode MUST NOT be executed under this identifier.

3.2 Surrounding audited draw-record metadata

The following values remain part of ScirDom's audited draw record, but they are not additional arguments to the public `execute_draw` reference APIs:

Metadata	Type	Description
`draw_mode`	String constant	MUST be `"unweighted_without_replacement"` for draws governed by this ASD.
`block_id`	String	Identifier of the active entropy block from which the draw-specific entropy bytes were taken.
`alg_spec_id`	String constant	MUST be `"scirdom.alg.v1.0"`.

4. Locked Participant-List Construction

Locked-list construction transforms parsed participant records into the deterministic execution-order register used by the ASD. Every conforming implementation MUST apply these steps in this exact order.

Step 1. Trim whitespace from `name`. Remove leading and trailing characters from the following explicit set only:

• Unicode General Category Zs code points: U+0020, U+00A0, U+1680, U+2000 through U+200A, U+202F, U+205F, U+3000

• Explicit additional characters: U+0009, U+000A, U+000B, U+000C, U+000D, U+001C, U+001D, U+001E, U+001F, U+0085, U+2028, U+2029

Python implementations MUST NOT rely on `str.strip()` for conformance. JavaScript implementations MUST NOT rely on `String.prototype.trim()` for conformance. Both languages MUST implement the explicit code-point set above.

Step 2. Remove empty entries. Remove any participant record whose `name` is the empty string after Step 1.

Step 3. Preserve order, IDs, and duplicates. The remaining records keep their parsed upload order exactly. Their `participant_id` values are preserved exactly. Duplicate names remain distinct records and are not collapsed.

Step 4. Assign indices. The locked register receives zero-indexed positions in preserved upload order. The first retained record is locked index 0.

Serialisation for hashing. Serialise each retained record as `participant_id<TAB>locked_name`. Join those lines with U+000A, with no trailing newline, then encode the result as UTF-8. This byte sequence is the locked-list serialisation used for hashing.

5. Duplicate Policy

Duplicate names are preserved.

The policy is exact and case-sensitive:

• `P-0001 / "Alice"` and `P-0002 / "Alice"` remain two distinct participant records.

• `P-0001 / "Alice"` and `P-0002 / " Alice "` remain two distinct raw records, but both become `"Alice"` after trimming and therefore remain two distinct locked records with different participant IDs.

• `P-0003 / "Bob"` and `P-0004 / "bob"` are distinct records.

• `P-0005 / "cafe■"` and `P-0006 / "café"` are distinct records. ASD v1.0 does not apply Unicode normalisation.

The participant count `N` used by the algorithm is the locked count after trimming and empty-entry removal. If those rules reduce `N` below the requested `K`, the draw is invalid.

Worked example:

Participant record	After trim	Retained?	Notes
`P-0001 / "Alice"`	`P-0001 / "Alice"`	Yes	Locked index 0
`P-0002 / " Alice "`	`P-0002 / "Alice"`	Yes	Locked index 1; duplicate name preserved
`P-0003 / "Bob"`	`P-0003 / "Bob"`	Yes	Locked index 2
`P-0004 / "bob"`	`P-0004 / "bob"`	Yes	Locked index 3; case-sensitive distinction preserved
`P-0005 / "Alice"`	`P-0005 / "Alice"`	Yes	Locked index 4; duplicate name preserved
`P-0006 / ""`	`P-0006 / ""`	No	Empty after trimming
`P-0007 / " "`	`P-0007 / ""`	No	Empty after trimming
`P-0008 / "Charlie"`	`P-0008 / "Charlie"`	Yes	Locked index 5

Locked result:

```
P-0001■Alice
P-0002■Alice
P-0003■Bob
P-0004■bob
P-0005■Alice
P-0008■Charlie
```

6. Entropy Consumption Model

Entropy is consumed from the active entropy block in strict sequential order.

Rule 1. Salt bytes are consumed first. Every draw consumes exactly 32 salt bytes beginning at `offset_start`.

Rule 2. Selection bytes are consumed second. Winner selection begins immediately after the salt bytes, at `offset_start + 32`.

Rule 3. Selection byte consumption is variable. This ASD deliberately uses exact rejection sampling. Therefore the number of selection bytes consumed is determined only at run time.

Rule 4. Rejected bytes count as consumed. If a candidate integer falls in the rejection region, the bytes used for that attempt are still consumed and MUST NOT be reused.

Rule 5. Salt bytes do not influence winner selection. The salt is used only for participant commitment hashing and offset accounting. The winner selection sequence depends only on the locked participant list and the selection bytes that begin after the salt.

Rule 6. Offset accounting.

• `offset_start` is the position of the first salt byte.

• `offset_end` is one past the final consumed byte.

• `bytes_consumed = offset_end - offset_start`.

• `salt_bytes = 32` always.

• `selection_bytes = bytes_consumed - 32`.

Rule 7. No fixed byte-count formula is normative. This ASD defines execution from an already committed `offset_start`. Any higher-level reservation or allocation policy belongs to the surrounding protocol and MUST NOT alter the exact rejection-sampling semantics in this ASD.

7. Salt Generation and Participant Commitment

Although the salt belongs primarily to the commitment layer rather than the winner-selection layer, it is defined here because it consumes entropy bytes and forms part of the reproducible evidence chain.

Salt length	exactly 32 bytes.

Salt source	the 32 bytes at positions `[offset_start, offset_start + 32)` in the entropy block.

Participant commitment computation:

1. Serialise the locked participant list as defined in Section 4.
2. Concatenate `salt_bytes || serialised_locked_list`.
3. Compute SHA-256 over that byte sequence.
4. Represent the result as a 64-character lowercase hexadecimal string.

This value is `participant_commitment_sha256`.

8. Uniform Integer Generation

To generate an exact uniform integer in the range `[0, N)`, where `N >= 1`:

Step 1. Determine the byte width. Let `byte_count` be the smallest positive integer `b` such that:

```
256^b >= N
```

Equivalently, `byte_count` is the minimum positive byte width whose state space can represent at least `N` distinct values.

Step 2. Read bytes. Read exactly `byte_count` bytes from the entropy source at the current selection offset.

Step 3. Interpret the bytes. Interpret the bytes as an unsigned big-endian integer `raw`.

Step 4. Compute the acceptance threshold.

```
max_val = 256^byte_count
threshold = floor(max_val / N) * N
```

`threshold` is the largest multiple of `N` that does not exceed `max_val`.

Step 5. Reject or accept.

• If `raw >= threshold`, reject the sample, count those bytes as consumed, advance the offset, and repeat from Step 2.

• If `raw < threshold`, accept and compute:

```
index = raw mod N
```

Return `index`.

This procedure is exact. It introduces no modulo bias.

Worked boundary examples

All examples below use the published test entropy source at byte offset 1000. At that location the first byte is `0x43` (decimal 67) and the second byte is `0xBE` (decimal 190).

N	byte_count	Bytes	raw	max_val	threshold	Accepted?	Result
1	1	43	67	256	256	Yes	0
2	1	43	67	256	256	Yes	1
5	1	43	67	256	255	Yes	2
255	1	43	67	256	255	Yes	67
256	1	43	67	256	256	Yes	67
257	2	43BE	17342	65536	65535	Yes	123
1000	2	43BE	17342	65536	65000	Yes	342

If this function is called with `N = 1`, it consumes one byte and returns 0. Under the direct-only ASD v1.0 contract, valid draws such as `K = N` may still reach this range-1 case during winner selection, and those bytes still count as consumed.

---

9. Exact Direct Winner-Selection Sequence Algorithm

9.1 Fairness target

For locked participant count `N` and requested winner count `K`, the algorithm MUST output an ordered sequence `W` of `K` distinct winners such that:

```
Pr(W = T) = 1 / P(N, K)
```

for every ordered `K`-tuple `T` of distinct locked indices.

This is the fairness target. Equal marginal chance for each participant is not enough by itself. The full ordered-sequence distribution must be uniform. If order is later ignored, the induced winner set remains uniform over all `K`-subsets.

9.2 Selection strategy

Let:

```
A = [0, 1, 2, ..., N-1]
```

The algorithm selects winners directly for exactly `K` steps. No complement mode exists under ASD v1.0.

9.3 Algorithm

```
Input: locked participant list of size N, requested winner count K
Let A = [0, 1, 2, ..., N-1]

For t from 0 to K-1:
    r = uniform_integer([0, N-t))
    j = t + r
    swap A[t] and A[j]

winner_indices_selection_order = A[0:K]
winner_indices_locked_order = sort(winner_indices_selection_order ascending)
```

The published winner identifiers are the locked entries at `winner_indices_selection_order`.

9.4 Output ordering rule

`winner_indices`, `winner_entries`, and `winner_identifiers` are the actual selection sequence. This order is semantically part of the result and MUST be preserved exactly by every conforming implementation, manifest writer,

certificate generator, and user interface.

`winner_indices_locked_order`, `winner_entries_locked_order`, and `winner_identifiers_locked_order` are derived audit conveniences only. They MUST NOT replace the certified result.

9.5 Proof sketch

At step `t`, the algorithm chooses one element uniformly from the `N-t` remaining elements. Therefore every ordered K-tuple of distinct participants has probability:

```
1 / (N (N-1) ... (N-K+1)) = 1 / P(N, K)
```

This is exact because Section 8 uses rejection sampling with no modulo bias.

If the recorded order is later ignored, each unordered K-subset corresponds to exactly `K!` ordered K-tuples, so the induced winner-set distribution is still:

```
K! / P(N, K) = 1 / C(N, K)
```

9.6 Fully traced two-winner example

This example uses the published vector `vector_3_ordered_two_winners`.

Raw entries	`['Eve', 'Charlie', 'Alice', 'Diana', 'Bob']`
Locked list	`['Eve', 'Charlie', 'Alice', 'Diana', 'Bob']`
Locked serialisation (hex)	`502d30303031094576650a502d303030320943686172 6c69650a502d303030330941 6c6963650a502d303030340944696 16e610a502d3030303509426f62`
Locked list SHA-256	`110fcd2be11a7591d21fcca406234e659a085fb2d7b77c8ed11db996260161a0`
offset_start	`201`
Salt	`85f8756472144c0db97e7c91e6a17a39635b47dd9dd9ad7f64eefb4ac45075ec`
Participant commitment	`07b47cc90a13c4537acdcad1e1b133c561db52783d61cb2a4264733cf981be23`

Step	t	Range	Byte	raw	threshold	Accepted	relative_index	j	Chosen locked index	Working array after step
0	0	[0, 5)	0D	13	255	Yes	3	3	3	[3, 1, 2, 0, 4]
1	1	[0, 4)	E8	232	256	Yes	0	1	1	[3, 1, 2, 0, 4]

Published winner indices in selection order: `[3, 1]`

Published winner identifiers in selection order: `['Diana', 'Charlie']`

Derived locked-order indices: `[1, 3]`

This example shows the required distinction clearly. The certified result is the selection sequence `[3, 1]`. The sorted locked-order view is derived afterwards and is not the primary result.

9.7 High-`K` direct-selection example

This example uses `vector_9_high_k_direct_selection`.

Raw entries	`['Eve', 'Charlie', 'Alice', 'Diana', 'Bob']`

Locked list	`['Eve', 'Charlie', 'Alice', 'Diana', 'Bob']`

offset_start	`700`

Step	t	Range	Byte	raw	threshold	Accepted	relative_index	j	Chosen locked index
0	0	`[0, 5)`	`DE`	222	255	Yes	2	2	2
1	1	`[0, 4)`	`33`	51	256	Yes	3	4	4
2	2	`[0, 3)`	`BC`	188	255	Yes	2	4	1
3	3	`[0, 2)`	`75`	117	256	Yes	1	4	0

Published winner indices in selection order: `[2, 4, 1, 0]`

Published winner identifiers in selection order: `['Alice', 'Bob', 'Charlie', 'Eve']`

Derived locked-order indices: `[0, 1, 2, 4]`

Even when `K = 4` and `N = 5`, ASD v1.0 still selects winners directly. There is no complement selection path.

---

10. Output Model

A conforming implementation MUST produce the following outputs for a valid draw:

The literal public `execute_draw` output contract is the table below. Any additional diagnostics, traces, or release-validation fields belong to supplemental tooling and MUST NOT be treated as extra public output fields.

Output	Type	Description		
`locked_entries`	List of records	Locked participant register in preserved upload order, with each item in the form `{participant_id, name}`		
`locked_count`	Integer	Locked participant count `N`		
`locked_serialised_hex`	String	Hex encoding of the locked UTF-8 serialisation		
`locked_list_sha256`	String	SHA-256 of the locked serialisation		
`salt_hex`	String	32-byte salt in lowercase hexadecimal		
`participant_commitment_sha256`	String	SHA-256 of `salt		locked_serialisation`
`winner_indices`	List of integers	Winner indices in exact selection order		

Output	Type	Description		
`winner_entries`	List of records	Winner records in exact selection order, with each item in the form `{participant_id, name}`		
`winner_identifiers`	List of strings	Winner `name` values derived from `winner_entries` for convenience only		
`winner_indices_locked_order`	List of integers	Derived locked-order winner indices for audit convenience		
`winner_entries_locked_order`	List of records	Derived locked-order winner records for audit convenience		
`winner_identifiers_locked_order`	List of strings	Derived locked-order winner names for audit convenience		
`offset_start`	Integer	First byte consumed		
`offset_end`	Integer	One past the final consumed byte		
`bytes_consumed`	Integer	`offset_end - offset_start`		
`salt_bytes`	Integer	Always 32		
`selection_bytes`	Integer	`bytes_consumed - 32`		
`k`	Integer	Requested winner count		
`mode`	String	Always `"unweighted_without_replacement"`		

Integration rule	Any caller, manifest writer, certificate generator, or user interface that displays the winner list produced by this ASD MUST treat `winner_entries` as the certified result. If a UI numbers the list for human readability, those numbers MUST NOT imply prize rank unless a separate draw rule assigns that meaning explicitly.

11. Edge and Failure Cases

These cases are normative.

Empty participant list after locked-list construction	HARD ERROR. No bytes are consumed and no result is produced.
`K = 0`, `K < 0`, or non-integer `K`	HARD ERROR.
`K > N` after locked-list construction	HARD ERROR.
`offset_start < 0` or non-integer `offset_start`	HARD ERROR.

Single participant, `N = 1`, `K = 1`	valid draw. The winner sequence is the sole participant. `selection_bytes = 1` under the current direct-selection contract, because one exact range-1 sample is still consumed after the 32-byte salt.
All participants win, `K = N`	valid draw. The winner sequence contains every locked participant exactly once. `selection_bytes` is not zero under ASD v1.0; execution still performs one exact direct-selection step per winner and consumes the corresponding rejection-sampled bytes.
Invalid UTF-8 input	pre-ASD ingestion error.
All entries empty after trim	HARD ERROR.
Insufficient entropy for the salt or for a later selection read	HARD ERROR.

12. Test Vectors

The published machine-readable vector file is `scirdom_asd_v1_vectors.json`.

The public vector file is a conformance bundle, not a draw-specific replay record. The entropy described in this section is synthetic test entropy used only for the published conformance vectors. It is not the draw-specific entropy disclosed in a completed draw evidence package.

Entropy seed	`scirdom-asd-v1.0-test-vectors`
Entropy length	`16384` bytes
Entropy generation rule	encode `entropy_seed` as UTF-8 to obtain the initial state. Repeatedly apply SHA-256 to the current state, append each 32-byte digest to the entropy stream in order, replace the current state with that digest, and stop once `entropy_length` bytes have been produced. Truncate the final digest so the published entropy source is exactly `entropy_length` bytes long.
First 64 bytes (hex)	`14c337a21e19f713838455e2fa0b6081baa654df0ebcb74a2515458edf00b3ab542f 8a9bd660d320acae7bfcec0932263cebfe8d5dbace1e9644365dbcc98bee`
SHA-256 of the entropy source	`fde3f338cc1ad2573609f50f3710f6bae727c95df6bb10a267b7fa0d652a7a2f`

12.1 Public vector schema

The public vector file has the following top-level shape:

Field	Type	Description
`asd_version`	String	Always `scirdom.alg.v1.0`
`entropy_seed`	String	Seed string for the deterministic synthetic test entropy source
`entropy_length`	Integer	Length in bytes of the synthetic test entropy source
`entropy_generation_rule`	String	Normative rule used to reconstruct the synthetic test entropy source
`entropy_first_64_hex`	String	First 64 bytes of the synthetic test entropy source in lowercase hexadecimal

Field	Type	Description
`entropy_sha256`	String	SHA-256 of the full synthetic test entropy source
`vectors`	Object mapping names to vector records	Published vector set

Each vector record has the following shape:

Field	Type	Description
`purpose`	String	Human-readable purpose of the vector
`inputs`	Object	Draw inputs for the published reference implementations
`expected_execute_draw_output`	Object	Literal public `execute_draw` output contract for that vector
`trace`	Object	Supplemental diagnostic trace material only

Each `inputs` object contains:

Field	Type	Description
`raw_entries`	List of records	Raw participant records for the vector
`offset_start`	Integer	Starting offset in the synthetic test entropy source
k	Integer	Requested winner count

`expected_execute_draw_output` MUST match the public Python and JavaScript `execute_draw` outputs exactly, including `selection_bytes`.

`trace` is supplemental release-validation material only. It may contain step-level diagnostics, rejection-sampling attempts, and other derived detail, but it is not the public API contract.

12.2 Vector summary table

Vector	Purpose	offset_start	N	K	Published winner indices	Derived locked-order indices	bytes_consumed
`vector_1_single_participant`	Single participant under direct-only semantics	0	1	1	`[0]`	`[0]`	33
`vector_2_two_participants`	Two participants; one direct winner selection	100	2	1	`[1]`	`[1]`	33
`vector_3_ordered_two_winners`	Two winners where selection order differs from derived locked order	201	5	2	`[3, 1]`	`[1, 3]`	34

Vector	Purpose	offset_start	N	K	Published winner indices	Derived locked-order indices	bytes_consumed
`vector_4_unicode_distinction`	Distinct Unicode spellings remain distinct in upload order	300	3	1	`[0]`	`[0]`	33
`vector_5_duplicate_preservation`	Whitespace trim with duplicate preservation and case sensitivity	400	6	2	`[0, 2]`	`[0, 2]`	34
`vector_6_multi_winner`	Three winners from ten; selection order differs from derived locked order	500	10	3	`[8, 0, 9]`	`[0, 8, 9]`	35
`vector_7_rejection_sampling`	Forced rejection and retry	3001	3	1	`[1]`	`[1]`	34
`vector_8_large_n_two_byte_range`	Two-byte range for `N = 300`	600	300	1	`[263]`	`[263]`	34
`vector_9_high_k_direct_selection`	High-K direct selection with no complement path	700	5	4	`[2, 4, 1, 0]`	`[0, 1, 2, 4]`	36

12.3 Rejection-sampling trace

`vector_7_rejection_sampling` demonstrates a rejected byte.

offset_start	`3001`

Salt	`ad0d98d0a51d7e9c84232433dcacd47245e099d074921f5aba2ce458d000b182`

Participant commitment	`9c4ef240efbb5a57a2b41f84ce97348afda1bd9e681127946387aacc5b3ef04d`

Attempt	Entropy offset	Byte	raw	threshold	Accepted?	Result
1	3033	FF	255	255	No	`null`
2	3034	3D	61	255	Yes	1

Published winner indices in selection order: `[1]`

Published winner identifiers in selection order: `['Yuri']`

The first byte FF lies in the rejection region for `N = 3` because the threshold is 255, so the byte is consumed and rejected. The next byte 3D is then consumed and accepted.

13. Verification Record

The published ASD bundle is supported by the following verification record.

Two guarantees are distinct and MUST NOT be conflated:

1. **Public conformance guarantee.** The five-file ASD bundle supports public conformance checking of the published reference implementations against the published synthetic test vectors.

2. **Draw-specific replay guarantee.** A completed draw evidence package, combined with the published ASD materials, supports independent replay of that specific draw. The draw-specific entropy for that replay comes from the evidence package, not from the public synthetic test vectors.

The internal release-validation runner and the internal parity harness are not part of the five-file public bundle.

1. **Reference-vector agreement.** The Python and JavaScript reference implementations MUST reproduce every published vector exactly.

2. **Parity-harness agreement.** The Python-generated parity case file and the JavaScript parity harness MUST agree on 10,000 deterministic generated cases.

3. **Disagreements block release.** Any disagreement blocks use of this ASD version until resolved by reference to this specification.

4. **Public conformance reproducibility.** Third parties MUST be able to verify conformance using only this document, the public vector JSON file, the public replay guide, and the published reference implementations.

5. **Draw-specific replay reproducibility.** Third parties MUST be able to replay a specific certified draw when given its completed evidence package together with the ASD publication referenced by that pack.

ASD v1.0 currently satisfies those requirements as follows:

• Public conformance bundle: the five downloadable public files are the specification PDF, the Python reference implementation, the JavaScript reference implementation, the machine-readable public vector file, and the public replay guide.

• Published vectors: 9 / 9 pass in JavaScript against the machine-readable vector file, with normative field checks passing against `expected_execute_draw_output`.

• Internal release-validation evidence: Python parity case generation produced 10,000 deterministic cases on 2026-04-19, comprising 8,271 valid cases and 1,729 expected-error cases, and the JavaScript parity harness passed all 10,000 against that case file with zero discrepancies.

• Draw-specific replay contract: completed draw evidence packs disclose the locked participant register, the draw-specific entropy bytes, the ASD identifier, the ASD document hash, and verification guidance sufficient for third-party replay.

14. Change Control

While the Phase 7 public beta remains pre-release, ASD v1.0 may still be corrected in place so the eventual live publication is internally consistent and first-principled.

Once ScirDom is live, any behavioural rule, normative statement, reference implementation, or published test vector change will require a new ASD version for future draws.

A later ASD version MUST NOT silently reinterpret historical draws.

15. Publication Record

Event	Date	Details
Initial authored version	2026-03-10	First complete written specification

Event	Date	Details
Upload-order execution contract corrected	2026-04-06	Locked participant list changed to preserved upload order with duplicate preservation, review-stage sealing, updated vectors, and updated reference implementations
Participant-ID execution contract corrected	2026-04-10	Locked participant basis changed from names-only lines to `participant_id<TAB>locked_name`, with structured winner records and regenerated public verification artefacts
Direct-only selection-sequence model finalised	2026-04-19	Complement mode removed, winner selection order made authoritative, and derived locked-order fields retained only as audit convenience
Machine-readable vectors regenerated	2026-04-19	9 published vectors updated to the direct-only selection-sequence contract
JavaScript vector runner passed	2026-04-19	9 / 9 vectors pass; 135 normative field checks pass
10,000-case parity harness passed	2026-04-19	10,000 / 10,000 cases pass; 8,271 valid cases; 1,729 expected-error cases
Bundle consistency correction issued	2026-04-19	Section 8 contradiction removed, public vectors split into `expected_execute_draw_output` plus `trace`, synthetic test entropy rule published, and conformance versus draw-replay guarantees clarified
Public PDF release issued	2026-04-19	Public specification and replay-guide bundle regenerated for the direct-only contract