`Object.getOwnPropertySymbols(value, options)`

Filtering own symbol properties by enumerability

Addressing a performance issue while working with symbols

Champion: `Ruben Bridgewater and Jordan Harband`

Author: `Ruben Bridgewater`

July 2025

1 · Motivation

- **Serializers / loggers** wish to include only enumerable data.
  Libraries attach metadata via **symbol-keyed** properties.
  ⇒ often need only *enumerable* **or** *non-enumerable* symbols
- Today's pattern
  i. `Object.getOwnPropertySymbols(obj)`
  ii. `filter()` + `Object.getOwnPropertyDescriptor` 🔍
- Drawbacks
  - Extra descriptor lookups (performance)
  - Boilerplate / error-prone
  - Engines cannot optimize

2 · Prior Art & Existing APIs

API	Scope	Enumerability filter?
`Object.getOwnPropertySymbols`	Symbols only	❌
`Object.getOwnPropertyNames`	String keys	❌
`Reflect.ownKeys`	Strings + symbols	❌
`Object.keys`	Enumerable string keys	✅ (strings only)

Many very big libraries (Lodash, Node.js, Angular, TypeScript, Next.js, and many more) filter for enumerability of symbols as described in Object.propertyCount.

2.1 · Real World use cases

- *Next.js*
- *Angular*
- *TypeScript*
- *Lodash*
- *React*
- ...

3 · Proposed API

```
Object.getOwnPropertySymbols(value, { enumerable });
```

option	type / default	result
`true`	boolean	**enumerable** symbols only
`false`	boolean	**non-enumerable** symbols only
`"all"`	string (default)	current behavior

`options` omitted/null → `"all"`.

4 · Illustrative Examples

```
const hidden  = Symbol('hidden');
const visible = Symbol('visible');

const o = {};
Object.defineProperty(o, hidden, { enumerable: false, value: 1 });
Object.defineProperty(o, visible, { enumerable: true,  value: 2 });

Object.getOwnPropertySymbols(o); // [hidden, visible]
Object.getOwnPropertySymbols(o, { enumerable: true }); // [visible]
Object.getOwnPropertySymbols(o, { enumerable: false }); // [hidden]
```

5 · Draft Specification Text (sketch)

1. Let *obj* be ? `ToObject(value)`.
2. Let *keys* ← ? `obj.[[OwnPropertyKeys]]()`.
3. Let *symbols* ← new empty List.
4. For each *key* of *keys*:
    - If *key* is a *Symbol* **and** `PassesFilter(key)`, append.
5. Return `CreateArrayFromList(symbols)`.

`PassesFilter(key)` uses `options.enumerable` (`"all"` | true | false).

6 · Alternatives

1. Separate methods (`Object.getOwnEnumerableSymbolProperties`, …)
2. Add filters to `Reflect.ownKeys` (string keys too)
3. **Status quo** – keep userland helpers

7 · Backwards Compatibility & Security and other Risks

- Non-breaking: 1-arg calls unchanged
- Two-arg form currently unused
- Polyfillable in ES5
- No new data exposed – only refined selection
- Accessor side-effects unchanged

8 · Open Questions

1. Allow explicit `"all"` or default only?
2. Future filters (`configurable` , `writable` , …)?

9 · Next Steps / Ask

- Address comments
- Advance to **Stage 1**
- Publish polyfill (`get-own-property-symbols`)
- Land spec PR + test262