# Documenting Invariants

TC39, July 2020

Yulia Startsev

# Current Status

- Some Invariants are documented directly in the specification.
- Undocumented Invariants are sometimes overlooked
- Some invariants have been proposed and contested
- So far, an Invariant has not required consensus in order to be adopted by the committee, they are informally applied.
- Invariants can and should change. The requirements for change should be written down.

# Idea: Collect and write down Invariants

- Will allow for knowledge sharing and the understanding of concerns
- Will ensure that cases where invariants are broken are clear and recognized
- Will ensure that even if a specific committee member is not present -- that the invariant is acknowledged.
- Invariants, if they are broken, should put the specification into a buggy state.  Invariants should be **Normative**.

# Definitions are loose right now

- **Invariant** can mean “a property of the specification that has been true up to the present”
- Invariant does not cover all guidelines
  - What about things we want to avoid?
  - What about properties of the spec which cannot be written down?
  - What about design concerns?

# For now...

- Let us consider “Invariant” to be **any protected property of the specification**

# Current Invariants

### 6.1.7.3 Invariants of the Essential Internal Methods

The Internal Methods of Objects of an ECMAScript engine must conform to the list of invariants specified below. Ordinary ECMAScript Objects as well as all standard exotic objects in this specification maintain these invariants. ECMAScript Proxy objects maintain these invariants by means of runtime checks on the result of traps invoked on the [[ProxyHandler]] object.

Any implementation provided exotic objects must also maintain these invariants for those objects. Violation of these invariants may cause ECMAScript code to have unpredictable behaviour and create security issues. However, violation of these invariants must never compromise the memory safety of an implementation.

An implementation must not allow these invariants to be circumvented in any manner such as by providing alternative interfaces that implement the functionality of the essential internal methods without enforcing their invariants.

**Definitions:**

- The *target* of an internal method is the object upon which the internal method is called.
- A target is *non-extensible* if it has been observed to return **false** from its [[IsExtensible]] internal method, or **true** from its [[PreventExtensions]] internal method.
- A *non-existent* property is a property that does not exist as an own property on a non-extensible target.
- All references to *SameValue* are according to the definition of the SameValue algorithm.

**[[SetPrototypeOf]] ( *V* )**

- The normal return type is Boolean.
- If target is non-extensible, [[SetPrototypeOf]] must return **false**, unless *V* is the SameValue as the target's observed [[GetPrototypeOf]] value.

**[[SetPrototypeOf]] ( *V* )**

- The normal return type is Boolean.
- If target is non-extensible, [[SetPrototypeOf]] must return **false**, unless *V* is the SameValue as the target's observed [[GetPrototypeOf]] value.

**[[SetPrototypeOf]] ( *V* )**

- The normal return type is Boolean.
- If target is non-extensible, [[SetPrototypeOf]] must return **false**, unless *V* is the SameValue as the target's observed [[GetPrototypeOf]] value.

**[[GetPrototypeOf]] ( )**

- The normal return type is either Object or Null.
- If target is non-extensible, and [[GetPrototypeOf]] returns a value *V*, then any future calls to [[GetPrototypeOf]] should return the SameValue as *V*.

**[[SetPrototypeOf]] ( *V* )**

- The normal return type is Boolean.
- If target is non-extensible, [[SetPrototypeOf]] must return **false**, unless *V* is the SameValue as the target's observed [[GetPrototypeOf]] value.

**[[GetPrototypeOf]] ( )**

- The normal return type is either Object or Null.
- If target is non-extensible, and [[GetPrototypeOf]] returns a value *V*, then any future calls to [[GetPrototypeOf]] should return the SameValue as *V*.

NOTE [[GetPrototypeOf]] for proxy objects enforces the following invariants:

- The result of [[GetPrototypeOf]] must be either an Object or **null**.
- If the target object is not extensible, [[GetPrototypeOf]] applied to the proxy object must return the same value as [[GetPrototypeOf]] applied to the proxy object's target object.

Alternative expression of an invariant

# Structural overview

# Structuring Invariants

Every invariant (section) should have:

- A description
  - Clear explanation of what the invariant is
- A set of definitions
  - Any key words that may be ambiguous must be defined
- A list of associated features of the invariant

# Structuring Invariants

Every invariant (section) should have:

- A description
  - Clear explanation of what the invariant is
- A set of definitions
  - Any key words that may be ambiguous must be defined
- A list of associated features of the invariant
- A rationale
  - Referenced when the validity of the invariant is questioned

# Invariant “Types”

Invariants should be **normative**. So far we have grouped them in these three categories.

- “Must/Must Not” invariants
  - Invariants that, if violated, put the specification into a buggy state
- “Should/Should Not” invariants
  - Invariants that have an “allow list” of cases where they can be broken
  - Otherwise, if these invariants are violated the specification is in a buggy state
- “Precedents”
  - Invariants that cannot be captured by specification text alone. A precedent references a decision made by the committee. (Example: Module resolution graph order and why it cannot be changed)

# Process for Proposing an Invariant

A few potential ways to handle this:

- Follow the same process as Normative Changes / Needs Consensus PRs
  - An invariant can be introduced more quickly.
  - This will make it easier to document existing ones that we have acted on
- Follow something closer to the Proposal Process
  - Will require multiple meetings to be adopted
  - More discussion time
- Other ideas welcome.

# Some Open Questions:

- Do these categories cover the types of meta-information used by the committee to make decisions?
- Are there any clear dangers here?
- How should this proposal be iterated upon?

# Ongoing work

- See documenting invariants
- Discussions have been taking place in SES

# Discussion

# Addendum: Clarifying the process