

# ~~Evaluators~~ new Global

## status update

*Zbyszek Tenerowicz (ZTZ) @naughtur.pl*

# Motivation

- Domain Specific Languages
- Test runners
- Principle of Least Authority
- Isolation of unreliable code

# Example - DSL

```
const dslGlobal = const new Global();  
dslGlobal.describe = () => {};  
dslGlobal.before = () => {};  
dslGlobal.after = () => {};  
  
const source = await import.source(entrypoint);  
await dslGlobal.eval('s => import(s)')(source);  
  
dslGlobal.document = mockDomApi;
```

## new Global

```
interface Global {  
  constructor({  
    keys?: string[],  
    importHook?: ImportHook,  
    importMetaHook?: ImportMetaHook,  
  })  
  // Unique to the new global:  
  Global: typeof Global,  
  eval: typeof eval,  
  Function: typeof Function,  
  // internal slots for *Function as well  
  
  // + properties copied from globalThis filtered by keys  
}
```

# Conceptual changes

- latest incarnation of the `Evaluators` proposal, from `Compartments` (Stage 1)
  - avoids adding new concept of `Evaluators`, reuses existing `Global` concept.
  - no new categories of global object, just replicas
  - host creates the global object
  - not opinionated on minimal set of globals
- “ `Global` picks up from the previous proposal for `Evaluators` and results from an observation that an object conveniently containing all evaluators already exists in the spec and all we need to do is expose a constructor for it.
- It also eliminates the concern where evaluators accepting any globalThis to use would clash with the host implementation's desire to use special objects only the host can create.
- No API to set a custom reference as global context for evaluators if it's not created via `new Global`
- When a new global is created it inherits all properties from parent global unless user specifies a list. Spec offers no opinions on minimal global, only demands that all evaluators are present.
- ”

## Details

- allows mutating `(new Global()).globalThis` before evaluation
- by default copy all properties from `globalThis`
- properties: `Global` and all evaluators have their internal slots relating them to the new *global*, that includes all `*Function` slots.

```
(async () => {}).constructor !==  
new Global().eval('async () => {}').constructor
```

## Details - All properties grafted by default

```
globalThis.x = {};  
const newGlobal = new globalThis.Global();  
newGlobal.Object === globalThis.Object;  
newGlobal.x === globalThis.x;
```

## Details - Properties can be selectively grafted

```
globalThis.x = {};  
globalThis.y = {};  
const newGlobal = new Global(  
  keys: ['y'],  
});  
newGlobal.x === undefined;  
newGlobal.y === globalThis.y
```



## Details - Some properties undeniable

```
const newGlobal = new Global({  
  keys: []  
});  
newGlobal.Object === globalThis.Object;
```

## Details - Own unique evaluators

```
const newGlobal = new Global();  
newGlobal.eval !== thisGlobal.eval;  
newGlobal.Global !== thisGlobal.Global;  
newGlobal.Function !== thisGlobal.Function;
```

## Details - Other unique intrinsic evaluators

```
const newGlobal = new Global();
newGlobal.eval("Object.getPrototypeOf(async () => {})") !==
  Object.getPrototypeOf(async () => {});
newGlobal.eval("Object.getPrototypeOf(function *() {})") !==
  Object.getPrototypeOf(function* () {});
newGlobal.eval("Object.getPrototypeOf(async function *() {})") !==
  Object.getPrototypeOf(async function* () {});
```

## Details - Inherits host import hook and module map by default

```
const newGlobal = new Global();  
const fs1 = await import("node:fs");  
const fs2 = await newGlobal.eval('import("node:fs")');  
fs1 === fs2; // if present
```

## Details - Can override import hook

```
const newGlobal = new Global({  
  async importHook(specifier) {  
    if (specifier === 'node:fs') {  
      return import.source('mock-fs.js');  
    } else {  
      return import.source(specifier);  
    }  
  }  
});  
const fs = await newGlobal.eval('import("node:fs")');
```

## Details - Closed holes

```
const fs = await (0, eval)('import("node:fs")');
```

```
const AsyncFunction = (async () => {}).constructor;  
const fs = await new AsyncFunction('return import("node:fs")');
```

```
const fs = await import(new ModuleSource(`  
  export default new Function('return import("node:fs")')();  
`));
```

Closing every escape gadget requires rigor — but is possible!

# Overlap with Module Harmony

```
const globalThat = new Global({
  importHook(specifier) {
    log(`global ${specifier}`);
    return new ModuleSource("");
  },
});
const source = new globalThat.ModuleSource(
  `
  import 'static-import';                // local static-import
  eval('import("direct-eval-import")');  // local direct-eval-import
  globalThis.eval('import("indirect-eval-import")'); // global indirect-eval-import
  new Function('return import("function-import")'); // global function-import
  `
  ,
  {
    importHook(specifier) {
      log(`local ${specifier}`);
    },
  }
);
await import(source);
```