WinterTC

Standards for server-side JS runtimes

Andreu Botella, Luca Casonato

Announcing the Web-interoperable Runtimes Community Group

May 9, 2022

Luca Casonato

https://deno.com/blog/announcing-wintercg

Goal

Make non-browser JavaScript runtimes more interoperable:

- Write for one runtime, run in any runtime
- Enable shared code between browser and server

Expand the common baseline of functionality available in all runtimes

W3C Community Group

- Started as a W3C CG
- Allowed open collaboration without membership requirements
- Unable to publish technical reports or standards

Need to publish a standard

Minimum Common API defines the common baseline functionality:

- `URL`
- `crypto`
- `fetch`
- `setTimeout`
- `ReadableStream`
- …

Looking for a new venue

- Last WEH we started looking for a new standardization venue
- Quickly found two options:
  - W3C WG
  - Ecma TC
- We had good discussions with both W3C and Ecma
- Major points of interest for us
  - Invited expert policy, to continue open collaboration
  - Simple organizational structure
  - Organizational help

Goodbye WinterCG, welcome WinterTC

January 10, 2025

Luca Casonato

https://deno.com/blog/wintertc

Ecma TC55

(Technical Committee #55)

a.k.a **WinterTC**

So what are we working on?

- **Minimum Common API**, defining the common subset of web APIs that should be available everywhere
  - **WinterTC test suite**, identifying a subset of WPT that matches the Minimum Common API
- **Sockets API**, to enable a unified API for TCP and TLS sockets everywhere
- **CLI API**, to enable unified access to env vars, args, cwd etc.

We are also working on upstream specs:

- **Fetch:** servers do not need browser specific security features (CORS, etc)
- **Web Crypto:** adding support for streaming data

Minimum Common API

- Superset of `[Exposed=*]` APIs
- Meaning many compute only APIs such as `URL` or `URLPattern`
- Additionally general purpose async I/O APIs like `fetch` or `setTimeout`

In the future: conformance levels

Minimum Common API is the minimum that everyone should support.

Additional possible targeted conformance levels:

- **Graphics**, with headless WebGPU, OffscreenCanvas, Image, DOMRect, etc.
- **CLI/File System**, including FS APIs, envs, args
- **Servers**, including advanced networking APIs (TCP/TLS/UDP/QUIC), and HTTP server APIs

Fetch

Most people know the `fetch` API:

```
const resp = await fetch("https://example.com");
const body = await resp.text();
console.log(resp.status, body);
```

On servers, Fetch is different though

- There is no current page
  - Thus no origin
  - Thus no cookie jar
  - Thus no referrer
- No need to protect against cross-site requests as there is no included-by-default authentication information
- No need for atomic redirect handling
  - Servers can be proxies, and they need to be able to see redirects happening

Default, but overridable Unforgeable

```
GET / HTTP/1.1
accept: */*
accept-encoding: gzip, deflate, br, zstd
accept-language: en-US,en;q=0.9
host: example.com
priority: u=1, i
referer: https://example.com/
sec-ch-ua: "Chromium";v="136", "Google Chrome";v="136", "Not.A/Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
sec-fetch-dest: empty
sec-fetch-mode: cors
sec-fetch-site: same-origin
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/136.0.0.0 Safari/537.3
```

Default, but overridable Unforgeable

```
GET / HTTP/1.1
accept: */*
accept-encoding: gzip, deflate, br, zstd
accept-language: en-US,en;q=0.9
host: example.com
priority: u=1, i
referer: https://example.com/
sec-ch-ua: "Chromium";v="136", "Google Chrome";v="136", "Not.A/Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
sec-fetch-dest: empty
sec-fetch-mode: cors
sec-fetch-site: same-origin
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/136.0.0.0 Safari/537.3
```

No default language

No origin, so no CORS, same-origin, referrer etc

Initial required changes to fetch spec

- Allowing various anti-spoofing headers for outgoing requests
- Allow cookie related headers in outgoing requests
- Allow reading various cookie related headers in incoming responses
- Fetch handling of `content-encoding` and `content-length` is broken, breaking proxies
- Request options such as `window`, `credentials`, or `referrerPolicy` are unused
- Response types such as `opaque`, `opaqueredirect`, etc are not needed

Some additional features are only needed by servers

- Configuring allowed protocols (h1/h2/h3)
- Programmatic mTLS
- Programmatic HTTP proxy configuration
- Custom CA certs
- Disabling connection pooling

We want to figure out a single extension point for `fetch` where servers can integrate these.

How do we want to do this?

Upstream!

We are not interested in a Fetch fork.

- Start with with a high-level description of needed changes
- Upstream maintenance work
  - To fix issues also relevant to browsers
  - To make selectively ignoring parts of the spec easier
- Investigate with Fetch editors the feasibility of "feature flagging" some parts of the specification

WinterTC test suite

A standard is worth little without a test suite.

The web platform has a test suite, WPT (Web Platform Tests).

- We are identifying the subset of WPT that tests the Minimum Common API
- The WinterTC test suite also should only rely on the Minimum Common API, so sometimes we need to modify tests
- As with fetch, the goal is to upstream our changes
- For additional levels on top of the Minimum Common API, we could add custom tests on top

Make our life easier when contributing to WPTs

- Do not use document-related APIs in tests unless absolutely necessary
  - e.g. only use `<form>` in `FormData` tests that test `new FormData(form)`

Needs document-related APIs	**Availability**	**File name**
Yes	Main thread	`<test>.html`
No	Main thread	`<test>.window.js`
No	Main thread and workers	`<test>.any.js`

https://wintertc.org

https://github.com/WinterTC55

https://min-common-api.proposal.wintertc.org

Meetings every two weeks (Thursday 16:30 CEST)
(must be an Ecma member / TC55 invited expert to attend)