Proposal Repo: https://github.com/tc39/proposal-async-context
Matrix Channel: https://matrix.to/#/%23tc39-async-context:matrix.org
Please reach out to @jridgewell or @legendecas for meeting invitation

Future Topics

- Short topics
    - Questions (for some future):
        - Any word from JSCore?
        - Is there a prototype for AsyncContext implementation which can be tested?
            - Non-up-to-date-AsyncContext implementations
                - https://chromium-review.googlesource.com/c/v8/v8/+/4553118
                - TaskAttribution is behind a flag (similar behavior, not identical)
            - Polyfill based on task attribution (only works on Chromium and requires command-line flags): https://gist.github.com/andreubotella/e061a42b17e4eefcd971aec5c396a9b4
            - Google's userland transform and polyfill: https://gist.github.com/shicks/0eec67c7b2d3693b6abf356d42a146a4
            - Ilias Bhallil polyfill: https://github.com/iliasbhal/simple-async-context
        - Are there any tests for the expectations being written? If so, where? As a tc39 proposal but with significant web integrations, what makes sense? WPT?
            - https://github.com/tc39/test262/pull/3874
                - Need test for module contexts
            - No WPT yet.
- Longer discussions

Oct 7, 2025

- https://github.com/tc39/proposal-async-context/pull/136
    - Nicolo: does the HTML spec say they could be in the same agent?
    - Andreu: the HTML spec says they're a different agent, chrome sometimes uses the same isolate
- Generator semantics
    - Steve: not doing anything in generators is more performant in browsers. Also more consistent on yield and yield* (also cf. userland iterator map() generator).
    - Nic: Kevin and SES group blocked the semantics. SES's call is on Wednesday.

        - Justin: Seems no delegate blocking understands the case that is breaking. https://github.com/tc39/proposal-async-context/issues/18#issuecomment-2010502039
        - Chengzhong: Original motivation was to allow mutation without a new function scope
        - Alternative API: https://github.com/tc39/proposal-async-context/blob/master/MUTATION-SCOPE.md#alternative-decouple-mutation-with-scopes
        - Justin: This could be acceptable. Mark does not like dynamic scope. This does not allow mutation without a reference to the AsyncContext.Variable instance.
        - Nic: A complementary way to preserve context on yield is adding a metaproperty like function.sent to read the context of the caller of next(). This only works directly inside the generator body. This has no performance cost as the context is already on the stack.
        - Chengzhong: the using proposal now only allows mutating a single Variable. Not a Snapshot. So that a manual invocation of the using symbol methods, skipping the @@dispose is limited to a single variable.
        - Next Action: An overview picture of all these alternatives.
- Generator semantics with async generators
    - Andreu: Nic's proposal of something like function.sent for the caller context, how does it work for async generators? If you call a `next` of an async generator without await, some calls of `function.sentContent` might be called after the caller of `.next()` has already exited that same context. But the promise returned by `.next` will still need to reference that context, because it's the context `unhandledrejection` will be called with if it rejects. So the context will be referenced anyway, just through more complicated means.
    - Not a problem per-se, but implementation-wise, the context of the next caller is always kept alive.
- Update of the design doc to Mozilla
    - https://gist.github.com/andreubotella/2325bd06cef379baefd011f0e6319148
- ~~Snapshot API alternatives?~~
    - ~~Context: https://github.com/tc39/proposal-async-context/issues/130~~
    - ~~AsyncContext.bind() a Function?~~
    - The idea is to remove the Snapshot API, just keep AsyncContext.bind()/AsyncContext.wrap().
- Michal: some people use generators and yield as an alternative to AsyncContext (?). Solid 2.0 might be working on something related.
    - https://effect.website/
    -

Sep 9, 2025

- WebPerf Call on September 11th: WebPerf WG Agenda

- Next meeting in two weeks is cancelled for TC39 plenary week.
- yield* context pass-through: https://github.com/nicolo-ribaudo/proposal-async-context/commit/32856cb7ce1aaaa9f310f8f4b6532b93b459012f
    - For context discussion: https://github.com/tc39/proposal-async-context/issues/18
    - This is a middle ground approach that allows `yield* iterator` to observe the calling context, while preserving the context around `yield` in a generator function.
    - When performing `yield* iterator`, the iterator can be implemented as a userland iterator with plain functions
    - Add this to the agenda of September Plenary.
- Pull requests:
    - https://github.com/tc39/proposal-async-context/pull/135
    - https://github.com/tc39/proposal-async-context/pull/98
        - Scott: Module as resource initiators (https://github.com/w3c/resource-timing/issues/263)
            - This PR only requires the module body evaluation to be in an empty context
            - Hosts implementing `import` can still observe the initiator context at the time of `import()` call
            - There is a referrer as a parameter in the host module import hook. Hosts can use that to infer resource initiators as well.
            - Scott: Import a style, and blocked on the style importing another CSS was added as <link>. Are there two contexts?
            - Jason (chat): There's also the potential for CSS Modules too right? `import someCSS with {type: 'css'; }`
- Gecko design doc: https://notes.igalia.com/EosN692sTN23Z-Iy0MElYQ
    - Andreu: This is still a draft, including a prototype impl.
    - For potential memory concerns around APIs like localStorage/BroadcastChannel, we may decide not to propagate at all.
    - Also notable about developer learning experience about why an API propagates in one condition and why it does not in another condition.
    - Planning to share this with Mozilla next week.
- Paint attribution tracking in Chromium
    - Andreu: Is this useful in task attribution?
    - Scott: for soft navigation, we want to know if an image change is caused by an interaction. Paint tracker links a paint to a soft navigation, and to an interaction.
    - Scott: right now it is just a subset of DOM modification, like adding a node to the DOM, source change of an image. Style change is not supported yet.
    - Andreu: no support with ResizeObserver
    - Michal: This is related to ContainerTiming. Like sets the AC.Variable whenever we observe a change to the DOM. In the future we can attribute a container change to an initiator.

Aug 26, 2025

- TPAC plan: https://www.w3.org/2025/11/TPAC/
  - Possible interested W3C WGs:
    - WebPerf WG (Pre-TPAC call is also valuable)
    - WHATWG Track
  - Planned schedule: 1 hour (30m presentation, 30m discussion or 1 hour discussion)
- Q: How does it work with Observales? And/or asyncIterable use cases
- https://github.com/tc39/proposal-async-context/issues/18
  - Nic: Can there be a middle solution between yield should be like await, and yield should not preserve context?
  - Nic: A possible solution can be pass-through the context of the `.next()` caller, to the inner generator of `yield*`, modelling this as a talk-through the inner generator when using `yield*`.
  - Example: `yield* gen()`, if gen() is implemented as a iterators, it can get the caller context

```
{
  [Symbol.iterator]() { return this; },
  next() {
    // In the caller context of .next().
    return { value: asyncVar.get(), done: true };
  },
};
```

Aug 12, 2025

- `dialog.after().then(() => ...)` expectation and behavior
  - Steve: Anything the async context proposal can do?
  - Justin: Second type of async variable, also about Stephen/Datadog's dispatch time context, but will we run into the opposite case?
  - Steve: the current workaround is to update the API to get rid of promise.
  - Chengzhong: is this an `after.then()` only issue? Does the same tracing library expect the two semantics at the same time?
    - https://github.com/WICG/observable?tab=readme-ov-file#concerns
  - Steve: second type of "continuation promise" or second type of async variable?
  - Justin: Attach dispatchContext to the event object, if the event object is passed to the then handler.

        - Chengzhong: `after()` can return a thenable, capturing the dispatch context
    - Transforming chains of `.then` to async functions
        - Steve: people expects this to be “equivalent” after the transform
        - Steve: Is there an “equivalent” transformation of user iterators and generator functions?
    - https://github.com/tc39/proposal-async-context/pull/132
        - Justin: in this PR, yield* still captures the context in the generator, and restores it after yield*.
        - Chengzhong: this is an editorial change if the next snippet behavior does not change.
        - Justin: does the PR defer the ctx 2 here to be (1), instead of (0)?
        - Andreu: the PR also captures the context at GeneratorStart, which preserve ctx (0). So this behavior does not change.

```js
function *gen() {
  // (ctx 2)
  const nextCtx = yield* yieldAndGetVar();
  // (ctx 2)
  // nextCtx => (3) can this be possible?
}

let g;
asyncVar.run((0), () => {
  g = gen();
});
asyncVar.run((1), () => {
  g.next();
});
asyncVar.run((3), () => {
  g.next();
});

function yieldAndGetVar(value, asyncVar) {
  let isFirst = true;
  return {
    [Symbol.iterator]() { return this; },
    next() {
      if (isFirst) {
        isFirst = false;
        return { value, done: false };
      }
      // return { value: new AC.Snapshot(), done: true};
      return { value: asyncVar.get(), done: true };
    },
```

```
  };
}
```

- Generator https://github.com/tc39/proposal-async-context/issues/18
    - Steve: What's the motivation behind generator semantics?
    - Chengzhong: https://github.com/tc39/proposal-async-context-disposable/?tab=readme-ov-file#limitations-of-run and https://www.npmjs.com/package/co
    - Steve: cases where people want to bring in the context of “next”
    - Steve: preserving the context across yield lost the flexibility in these cases
    - Justin: https://matrixlogs.bakkot.com/TC39_Delegates/2023-03-23#L173
    - Justin: it is still possible to wrap the generator, to pass the context of the next to pass to the generator.
    - Chengzhong: still skeptical about merging contexts on yield, like fallback to preserve if next didn’t provide a new abort signal in the context
    - Andreu: replay using variables after yield?

Jul 15, 2025

- https://github.com/tc39/proposal-async-context/pull/129
    - Steve: Is it distinguishable if the async context map is a WeakMap?
    - Nic: No. Having a WeakRef to a value in the async context map, it is observable if the value is never GC-ed. But if the value can be released, it is not observable.
    - Nic: The spec do not mandate the GC details.
- Firefox investigation:
    - MessageChannel:
        - One option is to only propagate when neither port has ever been transferred
            - Same as task attribution
    - BroadcastChannel:
        - There is a process-wide map keeping track of the number of listeners, with refcounting. The original thread is notified when all the other listeners handled the message, so it's possible to keep the context to propagate it and then do some clean-up there.
        - We also need to keep track of which thread a JSHandle belongs to, to know when to expose the context and when not.
    - IndexedDB and LocalStorage still wip
    - Will write down a document, maybe with attached patches on GitHub, and send it to Gecko
- WHATWG plans
    - Stakeholders: Domenic, Anne
    - More concerns on the spec side that every spec author will have to think about this.

        - Noise if the context is passed explicitly.
        - Hard to write tests (?) if being passed implicitly.
    - Spec notations like “queue a task”, “in parallel”, with questions like threads and cross-agents.
    - What’s actually breaking in incremental rolling out?
        - For tracing it’s an improvement
        - Frameworks depend on the context for dependency injection. This could be a breaking (when the context changes from A to B).
    - Can an implementation be rolled out incrementally, with the spec being a whole roll out?
- Jul 29 (in two weeks), the next meeting collides with the TC39 plenary and will be cancelled.
    - Andreu plans (no promise) to prepare the document before the plenary.
    - A proper time might be presenting at the Sept plenary, with Mozilla’s feedback on the document

Jul 1, 2025

- https://github.com/tc39/proposal-async-context/pull/126
- https://github.com/tc39/proposal-async-context/pull/127
- Firefox investigation:
    - MessageChannel
        - No distinction on message port if it is cross-agent. Firefox saves data like SharedArrayBuffer on a per-process map and passes ID across agents to detect if the data can be posted.
        - Nic: can we propagate only if the port has *never* been transferred? A port coming back is like a different port that goes _through_ the other thread.
        - Chengzhong: MessagePort is *meant* to be used cross-agent, we should maybe just never propagate the context and tell React to update their code. I wouldn't propagate the context in Node.js if I have an HTTP client and server in the same thread that talk to each other.
        - Chengzhong: also, right now there is no JS way to tell whether a messageport has been roundtripped or not.
        - Steve: it's a common pattern to use MessagePort as a fast scheduling thing. Jasmine for example also does it, not just React.
        - Chengzhong: Regardless, no support of propagation across roundtrips of the port.
    - BroadcastChannel
        - Same agent or different agents? Need to look more into this.
    - Chengzhong: Different principle? "Do not propagate for APIs that *could* be cross-agent". To avoid user surprise.
        - Nic: For a function that on different prototypes, like button.focus() and iframe.focus, should button.focus() propagate?

        - Nic: Databases that thought of as only single-agent, should their APIs propagate? Users might not be thinking about the potential cross-agent case.
    - Andreu: will look into conditions different tabs creating agents, and how it works with indexdb.
        - Nic: If this is implementation detail, they should be treated like always different agents.

Jun 17, 2025

- Feedback from Hackfest:
    - Notes: https://github.com/Igalia/webengineshackfest/wiki/2025-AsyncContext:-trimming-down-the-web-integration
    - Nic: smaug prefers a whole design. Asks about complexity to implement in firefox (potentially in webkit):
        - Propagate contexts on postMessage/BroadcastChannel across realms
        - Writing to IndexDB/LocalStorage and trigger events
    - Michal Mocny: is the question Firefox specific? Or is it “how easy is it to implement it?”
    - Shaseley: Any feedback from Webkit?
    - Nic: asked Anne in the past, but Anne has the HTML editor hat on.
    - Andreu: asked Suzuki (JSC, not WebKit), they were not fully looking into AsyncContext. Bun implemented AsyncLocalStorage. They were concerned about Bun’s use case.
    - Shaseley: WebKit position request https://github.com/WebKit/standards-positions.
    - Andreu: Bun(JSC)/Node.js(V8) did not implement yield support yet.
    - Nic: Did bun upstream the patch? (Andreu: no)
    - Michal: Relationship between Node.js ALS / AsyncContext
        - Chengzhong: General idea is to build ALS ontop of AC
        - Nic: API Parity, ALS.enterWith to be replaced with github.com/tc39/proposal-async-context-disposable
        - Chengzhong: https://github.com/nodejs/node/pull/58065 deprecating ALS.disable
        - Michal: any not supported use cases from ALS?
        - Nic: without enterWith, framework APIs like before/after hooks will need refactoring.
        - Michal: relying on Promise continuation to propagate?
    - Michal: negative was emphasis on shipping the full form of the proposal.
    - Nic: smaug worried about “a main page creates a realm/iframe, contains a AC.Var, calling to the outer page, and the main page captures the context and keeps the realm/iframe alive”

        - An option is to make context mapping a weakmap, performance concern was raised.

Jun 3, 2025

Cancelled, discussion will be on WebEngine Hackfest session, June 4: https://github.com/Igalia/webengineshackfest/issues/64

May 20, 2025

- May plenary: 2025-05 - AsyncContext web integration brainstorming
    - https://github.com/tc39/agendas/blob/main/2025/05.md UTC+2
    - Andreu: Frameworks feedback that they don't need particular events to propagate.
    - Steve: Important for users to not change most propagation cases. The API should do the right thing by default.
    - Shaseley: Are there some events still on the table to be supported? Like XMLHttpRequests
    - Shaseley: In our case, React will break if postMessage/MessageChannel/onmessage does not propagate
    - Justin: Registration context for event listeners can still be the reasonable option
    - Steve: Registration context is not useful in cases
    - Shaseley: lot of events run in empty context. Dispatch context makes sense
    - Justin: If dispatch context is possible, it will be good
    - Shaseley: Features like AsyncTask/async stack traces, Microsoft's effort on resource timing need to propagate context on events.
    - Shaseley: Mozilla has no position on task attribution (yet?)
    - Andreu: Dan Minor considered implementing scheduler.yield with the equivalent of CPED, without the rest of task attribution / AsyncContext.
- Web Engines Hackfest: https://github.com/Igalia/webengineshackfest/wiki AsyncContext: trimming down the web integration (tentatively June 3rd)
- https://github.com/nodejs/node/pull/58104
    - Set a value in CPED, during an async function before the first await, the mutation in CPED leaks out from the async function.

```js
async function foo() {
  setCPED(value);
  await 0;
}

const p = foo();
getCPED(); // value changed.
```

await p;
```
        - Chromium handles this by resetting the value.
        - CPED does not support restoring values in generator functions as well.
```js
function* gen() {
  setCPED(value);
  yield;
  getCPED(); // value lost
}
```
        - Justin: last with @Ron, with disposable to detect `using` syntax.
        - Shaseley: Not seeing issue with await change. Need to think about the generator's case more and talk about this next time.

Apr 8, 2025

- AsyncContext Stage 2 Update (slides)
    - New comment on https://github.com/mozilla/standards-positions/issues/1187#issuecomment-2786304707 suggests that we will need to go through both use cases and mental model complexities.
    - Andreu will present these slides.
- Disposable AsyncContext for Stage 1 (slides)
    - Clarify why we need a separate proposal
        - Dependency
        - The main proposal is valid by itself.
    - Realm boundary:
        - It's only "ShadowRealm Callable Boundary"
    - Composability
        - Signal compute function, using a variable.
        - Make it clear that all three solutions are compose-able.
        - Composing multiple variable mutations, does it fit into DisposableStack-like?
        - Before-each hook, can it return a value, and a test-framework can `use` it in the test body function.
    - @@enter with DisposableStack
        - Push @@enter to stage 2, reach out to @Ron
    - Slide 11 needs clarification on @@dispose
        - We should explain `using variable` carefully without saying that @@dispose will restore a snapshot, as "restore a snapshot" also affects other variables.
```

    - Potential prevention: fail fast to have .run methods to check not-disposed variable scopes.
    - Schedule the main proposal update before this new proposal.

Mar 25, 2025

- Events
    - Node.js Collab Summit https://github.com/openjs-foundation/summit/issues/446 April 1-2, 2025
    - BlinkOn 20 https://www.chromium.org/events/blinkon-20/ April 7-8, 2025
        - Public event.
        - Michal has a lightning talk about soft navigation. Recording will be shared publicly.
- https://github.com/tc39/proposal-async-context/pull/117 Merged
    - For Angular, mention signal and dependency injection.
    - Steve: they don't think it will be helpful, unfortunately.
    - Nic: Mozilla's standards positions say that the README focuses on frameworks, not benefiting end users. This document addresses this concern.
    - Michal: reading this doc shows the world with/without this feature.
- https://github.com/legendecas/proposal-async-context-disposable
    - Nic: Even the proposal can go with @@dispose only, we should push strongly with @@enter, to prevent users from accidentally mutating a value in the scope.
    - Nic: @@enter discouples the scope with the value being entered. So library can say "the variable will enter with this value", and then the user code actually exists it.

```
var1.run(100, () => {
  var2.run(200, () => {
        const x = var1.withValue(101);
        x[Symbol.enter]();
        var2.withValue(201)[Symbol.enter]();
        x[Symbol.exit]();
        // is var2 200 or 201 here? And var1?
  });
})
```

```
var1.run(100, () => {
  var2.withValue(201)[Symbol.enter]();
})
// what is var2 here?
```

- Nic/Steve: Currently run is implemented as one clone of the snapshot, with the updated variable, and then restore the previous pointer when it's done. If .run does not restore var2 in that example, it needs two clones instead.
    - Chengzhong: With Node.js AsyncLocalStorage, var2 would be 201.
- This addresses the concern that frameworks can not change their public API to adopt AsyncContext.run pattern.
    - Steve: Jasmine,
    - Chengzhong: and other test frameworks.
    - Shaseley: can we make it an error if the call order is messed up?
    - Chengzhong: we can define it before stage 2.
    - Nic: We still need to define the error state, because you can try/catch around the exit that throws
    - Andreu: Need to make sure it works with DisposableStack, if we make it an error
    - Steve: Ron is against about strict `using` declaration with @@enter. If this proposal allows loose manual invocation, this should not be an issue.
    - Nic: if we allow loose @@enter, we should make it hard to mess up.
    - Steve: how this is different from ContinuationFlow from messing up with the Variable one has access to
    - Michal (in chat): Is it possible to have a read-only reference to a variable?
    - Andreu: disposable across run boundary
    - Steve: curious about $ABA^{-1}B^{-1}$ cases:
        - const x = v.withValue(1); x[Symbol.enter](); const y = v.withValue(2); y[Symbol.enter](); x[Symbol.dispose](); y[Symbol.dispose]();
        - const x = v.withValue(1); x[Symbol.enter](); v.run(2, () => x[Symbol.dispose]());
        - const x = v.withValue(1); v.run(2, () => x[Symbol.enter]()); x[Symbol.dispose]();
    - Nic (chat): The reverse-wrap that Steve proposed could be AsyncContext.Snapshot.contain(() => runMaliciousCode())
    - Steve: bring up with SES confirming with the guarantees they were hoping.

Mar 11, 2025

- Mozilla's standards-positions: https://github.com/mozilla/standards-positions/issues/1187#issuecomment-2711498173
    - Shaseley: will share this internally, and to Shu
    - Steve: there are cases where it is impossible to pass the context explicitly, like the signal proposal handlers
        - Trend in app code doing loose coupling.
        - Event handler will issue an RPC, and a data store, and triggers a global state mutation, observations. There are no function called directly.
        - There are cases in Google that can not extend the argument list to pass the context.
        - TODO: Reach out to frameworks like Angular.

    - Nic: for users in frameworks (react, OpenTelemetry), write a passage for each of them about how the proposal can improve UX
    - Shaseley: how well is wording in spec like “queue a task” in implementation working out?
    - Andreu: APIs like XHR already need to track the API objects.
    - Shaseley: Storage API has not been looked at.
    - Nic: have a short list about implementation examples. And how task attribution implements them.
    - Shaseley: Mozilla might be concerned about “are we going to do this for EVERY SINGLE api?”
    - Nic: is Mozilla planning to implement Task Attribution?
    - Shaseley: no
    - Andreu: if async context is there, mozilla can implement task attribution easier.
    - Justin: Did we want to list real world use cases?
        - React async components being one
        - Examples of loose coupling where there is no direct function call to pass additional data through
            - Middleware (side effects registered to watch RPC responses)
            - RxJS
            - Google’s Signals that Steve talked about
        - Dependency injection (framework)
        - We have to think up framework changes that they may never make, before we can show how user code will benefit
        - Concurrent Server handlers would be a good one (My Vercel example from the original slide deck)
    - Nic: Vue's compiler also already polyfills AsyncContext :) https://github.com/vuejs/core/blob/fb0c3ca519f1fccf52049cd6b8db3a67a669afe9/packages/runtime-core/src/apiSetupHelpers.ts#L480
- Per-context vs. per-isolate state
    - Scott: is this a per-realm or per-isolate thing? On scheduler.yield, cases can be changing contexts in an isolate
    - Andreu: cross-origin?
    - Scott: embed frame A in frame B, a task in frame A triggers invocation in frame B.
    - Scott: AsyncContext makes it impossible to query other’s state, but it allows affecting them.
    - Andreu: It’s per-isolate. But I thought you couldn’t have different origins in a single isolate.
    - Scott: Same-site cross-origin code runs in the same isolate. Also cross-site in mobile devices.
    - Scott: Why is AsyncContext per-isolate?
    - Andreu: Stephen Belanger argued that it should be, matching AsyncLocalStorage’s behavior with Node.js vm module.
    - Andreu: cross-site interactions shouldn’t propagate the context

    - Nic: if a frame A calls B sync on the stack, is it a browser call, or a JS function call?
    - Scott: JS triggers an event in another frame, would have a frame A on the stack (?)
    - Scott: per-context propagation was slow, causing performance issues.
    - Justin (chat): When we discuss this with SES folks, we need to stress that this can be secured with the same mechanisms that we've already proven
    - Nic: what happens if A directly calls B's setTimeout
    - Scott: B's setTimeout captures B's context. Olli has concerns affecting each other's state.
    - Nic: if it's sync, isn't it good to be able to affect other's state?
    - Andreu: I don't think affecting the state of a same-site cross-origin iframe is an issue. For cross-site, it is. But cross-site same-agent interactions shouldn't be synchronous (e.g. postMessage), so not propagating the context shouldn't be an issue, right?
    - Scott: In Chrome there are synchronous cross-site interactions, e.g. calling `iframe.focus()` synchronously fires a `focus` event in the iframe. Same with `blur`.
        - https://github.com/whatwg/html/issues/3506
    - Chengzhong: Cross ShadowRealm sync function calls need to be able to propagate the context and this does not allow querying values without access to a variable instance.
    - Nic: it allows changing with a snapshot without access to the variable instance.
    - Andreu: If two sites happen to be in the same browser process / event loop / agent, they should behave like they are in separate processes.
    - Nic: in `focus` example, can you detect it's a sync call from the other iframe?
    - Andreu: you could observe it with timing side channels like Spectre, but not through anything intended by the specs
    - Nic: working on a principle on how the web propagates.
- Navigation behavior
    - Andreu: What will happen with navigation? Let's say we define a variable in a same-origin tab, and then we reload the page or navigate to a same origin url, can the context be observed after the navigation (through that variable)? What if we navigate an iframe instead?
    - Andreu: The current understanding is to only propagate across same-document navigation, never cross-document.
    - Scott: that's what we are doing now. Double check on same process navigation.
    - Andreu: For session history, there's the bfcache (back-forward cache), which keeps documents alive for some time after you navigate, to not have to reload them if you go back.
    - Andreu: But that can't be used to navigate back and forth synchronously. Can preserve context with a snapshot, but not *through* a navigation. So if a timeout triggers after a back-and-forth navigation, should it preserve context?

    - Scott: Task attribution probably does, because bfcache freezes and then unfreezes the event loop task queues.
    - Andreu: I don't think specifying this would be an issue, but it should definitely be tested.
- Actions
    - Chengzhong: Response to Mozilla's standards-positions
        - Nic: remember to link to React & Vue's document about AsyncContext.
        - Steve: will reach out to Angular ask to share publicly.
    - Scott: Check on task attribution on navigation.

Feb 25, 2025

- Memory Management Document: https://github.com/tc39/proposal-async-context/pull/115
    - Going to open a mozilla/standard-positions request after this PR lands.
- Prior Arts Elaboration: https://github.com/tc39/proposal-async-context/pull/116
- Continuation: https://github.com/tc39/proposal-async-context/issues/60
    - Support `await AsyncContext.Snapshot/AsyncContext.Variable write` for ergonomics.
    - Nicolò: https://hackmd.io/@nicolo-ribaudo/SJz-ydiqkl
    - There was discussions about `await` changing the context, documented in https://github.com/tc39/proposal-async-context/blob/master/CONTINUATION.md
    - Michal showed a demo like `await asyncVar.withValue(val)`. But with other Promises restore the context before `await`.
    - Mutation can be achieved with `.set` method, @@dispose only makes it harder to leak to the outer scope.
    - Steve: If this is left as a follow-up, how do these APIs work together?
    - Steve: FWIW, google internal tracing doesn't need continuation flow because we wrap separate children in their own spans and use mutable state within the span bookkeeping to flow context out

Feb 11, 2025

- Overflow from last time:
    - where should scheduler context propagation diverge from the async context propagation, like, preventing user snapshots from breaking soft navigation?
        - Shaseley: break soft navigation if user restores with `AsyncContext.Snapshot`.
        - Nic: Is this just about don't want others to change own's own variable

- Michal: Snapshot restores all variables. In this case, a special variable dont want to be changed.
- Chengzhong: does this only apply to specific scheduler API?
- Shaseley: a Snapshot.run breaks from the chain like `click`. Snapshot.run should only used by async state owners.
- Michal: should snapshot be limited
- Steve: callbacks in APIs like `setTimeout` need to be wrapped with snapshot.run
- Michal: any scheduling API, not only scheduler.posttask
- Justin: we do not expect anyone to touch async context, but only frameworks. Framework can use snapshot to continue soft navigation
- Michal: an internal value will be overridden.
- Michal: restore a default context in a user interaction event listener, loses browser internal variables
- Andreu: can be a host hook for browsers
- Nic: from react dev's pov, react owns the async state. Should security/permissions be affected by await / context.
- Michal: https://react.dev/reference/react/useTransition#react-doesnt-treat-my-state-update-after-await-as-a-transition
- Michal: `startTransition` is likely to be called in an event listener. And `setPage` initiates a softNavigation. Will `await someAsyncFunction` change the async state of `setPage`?
- Steve: No. `await` is not going to mess with the caller's context.
- Steve: Snapshots everything and it will be sure we are in the exact same snapshot, including browser states.
- Nic: Signal has some APIs for everything, and some APIs for framework. The names are different with the dangerous APIs. Maybe async context should do the same
- Michal: If user land scheduler is inevitable, this could be more helpful. Initial thought is that this can a simple break out of the jail.
- Andreu: better to not have Snapsnot to be Promise only.
- Justin: the context of promise uses the time of `promise.then`. Perform `promise.then` on a global promise does not inherit the global async state.
- Nic:

```
class ContextCapturer {
  #runner = setInterval(() => {
        this.#callbacks.forEach(cb => cb());
  })
  #callbacks;
```

```
  run(cb) {
        this.#callbacks.push(cb);
  }

  end() {
        clearInterval(this.#runner);
  }
}
```

- This allows "stealing" the context without Snapshot.
- It captures when you do `new CaptureContext()`, and runs the callbacks in it
- Michal: might not trust a userland scheduler to associate the soft navigation. Some value in userland scheduler.
- Justin:

```
const nilCtxQueue = [];
let nilCtxPromise = Promise.resolve().then(run);

function run() {
  nilCtxPromise = Promise.resolve().then(run);
  const queue = nilCtxQueue.slice();
  nilCtxQueue.length = 0;

  for (const item of queue) {
        item();
  }
}

function scheudleInNil(cb) {
  nilCtxQueue.push(cb);
}
```

- It is possible to propagate nil context just with promise.then
- Andreu: I will not oppose to have host hook to preserve browser internal variables on snapshot/restore. The host hook would take the context to restore, and return either an alternative context to restore, or empty. And the default host hook is just returning empty.
- Michal: BTW I just found https://github.com/tc39/proposal-async-context/issues/85 is related to previous conversation?
- Nic: yes

- https://github.com/tc39/proposal-async-context/pull/111

- No updates on next week's TC39 plenary

    - Batching with HTML integration feedback addressed.
- Next time:
    - Feedbacks&document on memory management
    - Michal: https://github.com/tc39/proposal-async-context/issues/60
    - Michal: Support `await AsyncContext.Snapshot/AsyncContext.Variable write` for ergonomics.

Jan 28, 2025

- WHATNOT feedbacks:
    - Handling should be uniform for everything
    - Semantics should be worked out in advance
        - List events that should use registration context, and all the rest use dispatch context
        - Events dispatched synchronously, are always in dispatch context
        - User initiated events do not propagate context, and use fallback context
        - Events triggered by API but not synchronously, like XHR, `postMessage` in the same agent
            - Michal: A very common pattern to effectively yield
            - Polyfills based on same window postMessage, may need propagation
            - An alternative API might be needed for polyfills.
            - Michal: for curiosity I checked core-js and it will use MessageChannel whenever available (https://github.com/zloirock/core-js/blob/master/packages/core-js/internals/task.js#L80-L86) for setImmediate
            - Capture a Snapshot can be an alternative
        - Scott: Have FS operations, IndexDB been covered? Devtools instruments async stack traces, there is something missing (?)
            - Do a lot “queue a task and fire an event”
            - For tracing libraries, the ability to get the context at platform API call is sufficient for instrumentations.
    - We shouldn't save the context implicitly [except for Promises?] to avoid leaking memory
        - The leak is hypothetical, can this be an opt-in?
        - Andreu: The feedback was more like if a context can be alive for indefinite time
        - Implicit propagation is the core of the proposal
        - Andreu: for like `setTimeout`, the context is save along with the callback. When it times out, the references to the callback and context is cleared.

        - Window1 created a setTimeout, and a callback is from another window. When window1 is collected, the callback is still scheduled (?).
        - Another design is making the context map a weak map.
        - APIs like `requestIdleCallback` may never call the callback if there is no idle time.
        - When will an AsyncContext.Variable be collected, and making the entry in the global context weak map expire?
            - E.g. when an iframe/window is no longer referenced, the instances can be collected, and its entries in the global context can be collected as well.
            - Generally, an AsyncContext.Variable is created at a module-top-level scope, and will not be collected, unless the whole window is destroyed.
    - The last call was short of time and the topic was out of time.
    - The memory leak concern needs to be clarified if only certain APIs like Observers, or all APIs should not implicitly propagate.
    - Next time:
        - where should scheduler context propagation diverge from the async context propagation, like, preventing user snapshots from breaking soft navigation?
        - https://github.com/tc39/proposal-async-context/pull/111

Jan 14, 2025

- https://github.com/tc39/proposal-async-context/pull/100 was merged
    - Andreu: If a stream is transferred to another agent and be transferred back, should there be any context propagation or preservation?
    - Nicolo: postMessage to yourself. Trace across workers would be nice, but requires agent-wide GC. For request-response worker interaction, better to have a separate promise-returning API.
    - Chengzhong: real world use cases are helpful.
- #107: Event.captureFallbackContext follow-up
    - Nicolo: should a “read” fallback context API be exposed? The current shape only allow writes the “fallback context”, but not allow reads what the “fallback context” contains.
    - Steve: the point of the fallback context is that the code is not aware of the fallback context. “Read” breaks this assumption.
    - Nicolo: the idea would be to set a new updated fallback in the registration-variable's run method
    - Steve: this would break the fallback for other variables that didn't want this behavior.
- WHATNOT: https://github.com/whatwg/html/issues/10906

- A polyfill based on task attribution: https://gist.github.com/andreubotella/e061a42b17e4eefcd971aec5c396a9b4

Dec 17, 2024

- Longer discussions
  - https://github.com/tc39/proposal-async-context/pull/100
    - Can we merge and iterate on it?
      - Add clear notes about TODO points.
    - Continuation variable allows “returning” value around “await”s.
    - Dispatch context on events will not allow this.
    - https://github.com/tc39/proposal-async-context/issues/107

Nov 26, 2024

- Longer discussions
  - Web Integration #100
    - Proposed solution:
      - Start with a set of asynchronous events, use the dispatch context. Browser events use the “fallback” context.
      - For synchronous events, use the context where the events were dispatched.
    - Definitions:
      - Sync events: dispatched in the middle of a task (i.e. immediately when triggered), i.e. abortSignal
      - Async events: a new task scheduled, new entry point, but were triggered from a previous task, i.e. setTimeout, XHR.send, postmessage. Most events?
      - Browser events: new entry point, triggered without a previous task, start with empty context, i.e. user interactions or os events, click, scroll…
  - Discuss: js module execution
    - adding script, or dynamic import(), ShadowRealm.prototype.importValue()
    - We don’t evaluate modules more than once. Multiple callers might request/add the same module. So we don’t want to set the AsyncContext based on the caller, instead some global?
  - Task Attribution V2
    - APIs like requestAnimationFrame doesn’t queue a task and run callbacks at once
  - async/await and mutation inside a function scope: #101
- Short topics
  - #98: should the host hook be limited from accessing the importer’s async context snapshot?
    - NIc: Consistent root context per realm.

        - ShadowRealm
- Next actions:
    - Update #100 to the latest proposed solution.
    - switch on/off weeks for meetings (other bi-weekly) starting Dec 17.
    - shaseley@google.com, mmocny@google.com

Nov 12, 2024

- Short topics
    - Browser feedback on web integration
        - Mozilla is implementing scheduler.yield. Scheduler.yield only propagates through promise and async/await, nothing else. No web APIs integration. Separate implementation.
            - Changes how snapshot interacts with scheduler and/or task attribution
            - Also determines whether promise or setTimeout is more expensive
                - though this depends on details of how maps are implemented, can keep some variables in a separate bucket that does/n't propagate under certain circumstances
        - Incremental addition of async context on events. List the initial set of events integrate async context, e.g. unhandledrejection, postmessage
        - Prefers a simple solution of fallback context. clarify motivation of fallback context trees
            - specifically, only falling back when browser directly triggers the callback without a dispatch context (n.b. if dispatch context is specifically set to root context, it would not fallback)
            - does it ever make sense to use the same fallback for a non-event?
    - Iterator helpers
        - iterator helpers are a lot like sync events - shouldn't trigger fallback
        - also an analogue to generators, which run in initial context
            - may want to rethink generators and treat them more like sync events?
                - more expressive, easier transpilation
                - generators also currently need extra special handling in the spec, don't just fall out from promise/await
            - assumption made with using syntax was that context will be restored inside generator body - treating as sync event would make that future extension problematic
            - weakens idea of context being block-scoped - less teachable

            - can we work around these counterpoints? how important is block scoping?
                - may be more open to something like enterWith?
    - Snapshot identity
        - Angular: am I in the current snapshot? be more performant
        - Cross-realm identity
        - If the bootstrap snapshot is a normal variable, and reference the variable in context it is not bootstrapping, how would it tell it is in bootstrap
        - This mattered in a tight loop in the framework's internal event queue - making new snapshots every time was a non-starter
            - wrap() might help a bit if it can be more performant than Snapshot()?
            - Dan had some ideas of how to avoid making a bunch of objects
- Longer discussions
- Next actions:
    - Invite scheduler API folks to this meeting
    - Plenary in 3 weeks, switch on/off weeks for meetings (other bi-weekly) starting next week
    - Merge soon PRs:
        - https://github.com/tc39/proposal-async-context/pull/100 trim down events part
        - https://github.com/tc39/proposal-async-context/pull/98

Oct 29, 2024

- Short topics
    - https://github.com/tc39/proposal-async-context/pull/106
        - How often will people need to set multiple variables at once?
            - generally we don't expect it to be often
            - a few examples of v8 internals (scheduler, etc) needing to set multiple, but individual libraries will generally not
            - batching is always possible, it depends on how often variables change together whether this is more ergonomic/efficient or not - if variables often change together, batching is a win; if not then it's also less likely to need to change both at the same time - userland libraries should be feasible in this case.
        - Does this proposal allow specifying a default value?
            - could put it on the get() call, but this shifts burden of knowing the correct default to the caller.
        - The static run/get example only looks better because we're setting two vars - if setting one, then the current proposal is still preferable
        - A disposable-based approach also obviates the need for a multiple-set API

        - Steve H is concerned about polyfillability if this is a later proposal
    - Andreu is coming around a bit to the mutation root example (suggested in #101)
        - This is a possible solution to testing issues
        - Might be difficult to polyfill
        - Could also look into a copy-on-write solution where mutations don't affect earlier uses
    - Conclusion: we can probably link #105/#106 to #101
    - We should also start considering how quickly we can land the extensions once we hit 2.7
- Web integration
    - Is a stack needed?
        - stack "feels" better but isn't strictly needed, we can just special-case the context that comes from the browser's event loop - seems to satisfy all known concrete use cases
        - original proposal marked out zones of capturable context, all event listeners triggered from within will use a context from within; Justin proposed this is too complex and that instead we just guarantee that browser-triggered listeners will run in the fallback context
        - original proposal allowed two different areas to call back and forth and get the containment context rather than seeing each other
    - Is it reasonable to piggyback other things on this, not just events?
        - are there other callbacks dispatched directly from browser?
            - schedulers (setTimeout, etc) - but these use registration context
            - media events have races between causes
            - streams' constructor methods can be called directly, e.g. if a readable stream passed to fetch
            - more convoluted cases, e.g. stream transformers' ctor methods calling into readable streams
            - geolocation APIs (not events, dispatched async similar to setInterval)
            - userland APIs
            - worklets - action comes from different agent - method takes a function/class and calls methods from a different agent - similar to workers so maybe we don't expect context to flow at all?
                - workers have their own event loop and to keep going, but worklets are short-lived
        - event-based API means we don't need to explicitly wrap the callbacks, since addEventListener does it automatically (though it still can even if the wrap function is explicit)
            - motivation was that events triggered by user action don't have an obvious context
            - would be nice if anybody can emulate browser context, rather than having a special privileged context

        - Frameworks (like signals) able to delimit the parts of a program, not only events.
        - passing functions still allows leaking - too expensive to capture fallback on function instantiation, so best only used between major separated islands in app
    - what about dynamic import?
        - currently uses root context since it could be imported from multiple places - allows escaping the fallback, so no guarantees about containment: `containFallback(() => import(...))` - if imported module registers an event listener, then it ends up outside the container
        - Task attribution does something for module loading/fetching - potentially not just propagating the root, same-origin iframes may be different but not clear on details
            - should reach out to Scott Haseley about this
    - Nic: What happens if `.addEventListener` a single listener to different containments
        - if fallback is part of key then it's impossible to unregister once you're outside the fallback, since you can never get back into it, so probably better to just use the first one and make future calls no-op
- AIs
    - Andreu: investigating iterator helpers
    - Andreu: investigate task attribution + modules
    - Nicolo: talk w/ Mozilla about proposed browser integration
    - Steve: working on containment stack polyfill proof of concept
    - Chengzhong: continuing mutation explainer, merging #106 into #101

Oct 15, 2024

- Web integration proposal https://hackmd.io/@nicolo-ribaudo/rJhdk8HyJe
    - Can the containment stack be simplified? Need concrete case to explain the complexity
    - Other scheduling mechanisms like Observables that batches observations
    - Steve: frameworks want confidence that their contexts die
    - Justin: does the fallback tree proposal mean that bootstrap contexts will live forever?
    - Steve: is the fallback context concept broader than just events - do we want non-event schedulers (e.g. observers, lifecycle callbacks, signals, iterator helpers, streams, etc) to use the same fallback root instead of registration or global root in some/all cases?
- Update on Oct Plenary
    - SES model on browser internal variables, expanding the concerns.

    - SpiderMonkey wants AsyncContext in replacement of incumbent realm implementation.

Oct 1, 2024

- TPAC feedback
    - Web editors prefer registration context.
    - If queue task in parallel can work in implementation with strategy other than registration context, web editors would not oppose the solution
    - Q: what's parallel queue task
        - HTML queue events from different threads in parallel
        - Difficulties for web editors to determine the "right" context
        - In spec, modify the phrase "Queue a task" to also take a snapshot and restore it at task run. In impl, work can be either in another agent, another thread, or another process. Difficulties can be in the cross-process communication to reference the snapshot.
    - Currently Task Attribution uses dispatch context, but may be open to changing to registration context (TODO: Ref), will run a finch trial to see the impact of the switch, but caveat that Task Attribution may not be in the best state for the trial
        - for all but the few they specifically use dispatch for - message, popstate (history events), xhr (?)
        - TA uses CPED in v8, we expect to build on top of that; but scheduler.yield also uses CPED except that its state gets cleared by setTimeout and await (i.e. different behavior than AsyncContext will need) - (link) considers setTimeout as a new task, only propagate signal/priority for microtasks/await
            - setTimeout just needs to do an extra step to reset the TaskSignal (or other internal vars) when it runs its task, can be done without significant performance regression
    - Next step: if it is implementable in other browsers.
- Web Integration Docs
    - Dispatch & registration context and enhancements: https://hackmd.io/@nicolo-ribaudo/ByPl9jF6A
    - Web integration in-spec context tracking: https://hackmd.io/5Esc8zq8SJSrettCSKTVag
    - Example of complex system communicating through events: https://hackmd.io/02hmitJLTXe7Us9M7Lh2YQ?view
    - Example of how multi-app event handling: https://gist.github.com/nicolo-ribaudo/c16386e211313edabda9680b434f7cea
- Next: Steve will share examples and matrices.
- Andreu is going to add an agenda item about Web Integration in Oct Plenary.

Sep 3, 2024

- Web Integration
    - Updates:
        - Added a column in DOM Events (NOTE: SHARED EXTERNALLY) “Context easily available at async dispatch”
        - HTML spec editors don’t want web spec authors to have to think about which context each event should have. We’re looking at whether the context could be propagated automatically with HTML spec algorithms. Trivial for 50% of events, but not so much for the rest, because some specs (e.g. CSS) are very hand-wavy (declarative rather than imperative). Also not easy for fetch, since it calls into the HTTP RFC, and it’s not clear how the context flows in non-web specs. This is an analysis on spec complexity, not implementation complexity.
        - Implementations are imperative, not declarative, so those cases would be easier. But in many cases browsers might be optimizing and not having the same behavior. Would need a lot of WPT tests.
    - Chengzhong: Publish a gist for a document categories events used in OpenTelemetry like singleton event listeners and non-singleton event listeners.
- Module top level context: https://github.com/tc39/proposal-async-context/pull/98
    - “Module and script records could snapshot the context where they run, and run module.Evaluate() in dynamic import using that context”
    - Create an internal slot SourceTextModule[[AsyncContextMapping]] and script records, and not make this host defined.
    - Host can set the host defined context first, and create/evaluate the root module. No need for a new host hook.

Aug 20, 2024

- Attendees: Jatin, Andreu, Chengzhong, Dan, Justin, Jeremy Shaker, Steve Hicks, Ariel Backenroth
- Web integration
    - Proposal: Registration time always https://github.com/tc39/proposal-async-context/pull/100
    - Steve: You want the most relevant causal context in any callback. The most recent JavaScript that ran. You have several different JS contexts, in general, which contributed to something occurring.
        - DOM Events (NOTE: SHARED EXTERNALLY)
    - Steve: We're trying to build a new tracing system from scratch, on top of AsyncContext. Registration context is working out to be infeasible. If it's from user interaction in the webpage, there's no JS that triggered it. But if you dispatch a click event manually, we could allow that to run. There are all kinds of things in

DOM which have causal contexts, e.g., innerHTML or deleting an element from JS triggering scheduling a MutationObserver.

- Chengzhong: OpenTelemetry instruments event handlers in a way that's specific to the particular event, e.g., they can pick the expected context when there are multiple contexts available. If they are in a manually triggered event vs user interactive.
- https://github.com/open-telemetry/opentelemetry-js-contrib/blob/main/plugins/web/opentelemetry-instrumentation-user-interaction/src/instrumentation.ts#L305
- Andreu: Frame the difference between setTimeout and user interactions in terms of, not registration, but what starts the async thing which causes the event. For setTimeout, logic could be based around starting the timer, for a user click there's nothing started from JS.
- Dan: Can we learn from what OTel does? Didn't that solve this problem on the basis of registration time, via Zone.js?
- Chengzhong: OTel patches addEventListener.
- Jatin: We want to not have to have any kind of runtime library, and have the browser just do the correct thing, rather than needing to patch addEventListener.
- Chengzhong: Different tracing libraries may choose different things
- Justin: d3 isn't going to know about tracing, so it seems like it shouldn't make a difference between initial vs registration, since the tracing library can set up the registration context suitably before calling into d3
-
- Ariel: it's important to know when a trace is done - having the registration context around much longer means that it continues to be accessible and may never be done; w.r.t. OTel, this seems to be a known limitation.
- Dan: Knowing when a trace is done - Yoav Weiss looked into this for task attribution, and found this to be very complicated and ultimately infeasible. How do other platforms/systems handle this?
- Ariel: In android we use GC, in iOS we use reference counting
- Justin: curious about bootstrapping and knowing that it's done; it sounds like you're trying to use GC as the signal, and the only way to get this to work is to avoid having any references leftover; this seems wrong because (1) you can't rely on GC timing, and (2) an "end" callback should be suitable to indicate when it's done.
    - Ariel: we don't rely on GC at all for web tracing (the WeakRef/FinalizationReg spec says not to)
    - Steve: We're conflating these things because with registration context, the bootstrap context keeps coming back. It's not that we're trying to get a signal of when it completes, it's that once it's done we don't want it coming back.
    - Ariel: We do use reference counting, so the more we have that keeps the RC alive, the … But to Steve's point, once we've completed bootstrap we dispose of stuff. We also use that to time the bootstrap, with the timing relying on reference counting.

    - Steve: But knowing when it's complete and when to ship it…
    - Chengzhong: echo Ariel's point that OTel has these limitations - the API to end a span is explicitly separate from removing span from the context; even when a span is ended, we can still extend the causal children by looking into the async var to get a span graph with child after parent is finished
    - Yoav's previous task termination investigation: Task termination (Note: The backing task attribution mechanism has changed since this doc was written)
    - Ariel: Not just a web limitation, Android has similar issues when knowing whether a trace is active, with everything happening in just one thread. The key bit of information is who called me.
    - Dan: This is a long-lived FR - Node's async-hooks had dispose hooks, and it was a big challenge when migrating to native promises, always just kind of worked poorly, and we've been moving away from making this functionality available (e.g. with AsyncLocalStorage, which replaces async-hooks but doesn't solve this problem).
    - Task termination issue thread: https://github.com/tc39/proposal-async-context/issues/52
    - Chengzhong: Termination and async variables are separate, can use `using` to handle termination, though not perfectly
- Andreu: Chrome has some code to track source of events (surprisingly unrelated to task attribution). Trying to print this source whenever an event is fired, whether it's user or browser initiated. Not sure how to get the data, could run all the unit tests and web platform tests - could get all the data and munge it, but some issues with sandboxing and not being able to get the results out of the sandbox. Not 100% confident about the hacks, these APIs weren't intended to be used this way, sometimes there's a whole chain of algorithms leading to an event, may not have entire sequence tested
- Steve: Many events have multiple types of sources: user interaction and programmatic - this means they can already have different types of contexts, so it's unlikely people will depend on one specific type
- Andreu: Will be at TPAC next month (W3c event) - can discuss w/ web platform folks there.
    - Andreu: Domenic had expected (on #100) more of the web integration to be automatic via WebIDL - the manual data flow propagation for dispatch contexts was already too much, and what we're looking at now will be even more
    - Steve: many specs just say "queue a task to … fire an event", and we probably just want to search-replace to "let S be current snapshot, queue a task to … fire an event with snapshot S"
    - Andreu: Parallel algorithms make this more complicated, where the spec text essentially spawns a thread and then post a task to the event loop. This isn't always implemented straightforwardly in browsers.

- Dan: this is the core issue, parallel algos would need to save snapshots; need a very clear explanation of why this is useful

Aug 6, 2024

- Short topics
- Longer discussions
    - Web integration #100
        - Jeremie: Integrating tracing in angular - trying to track interactions; when a user clicks a button, triggers an ngrx action, ngrx uses setInterval - registration context is when store is defined
        - Andreu: to clarify, library handles the listener, event bubbles out to user-defined context
        - Andreu: one reason for not exposing the dispatch context is that synchronous events would need to be tracked specifically in the spec
        - Steve: If we do nothing in the spec, don't we just automatically get the dispatch context for sync events?
        - Andreu: yes, for sync events, but not for async events - we'd get the top-level
        - Dan: Only two options - either sometimes have a contextual context and sometimes either empty or registration; or else always registration
        - Andreu: currently no good way to check if a given context is the empty context, whereas when it's an event property, it could be null
        - Dan: The failure mode with the former is that you're just missing context, that could be okay?
        - Dan: Steve should enumerate which events should run in which context, Andreu can help with enumeration
        - Steve: Click event running in registration context by default seems reasonable?
        - Peter: Does the dispatcher maybe have the ability to know what context it wanted and use that?
        - Jatin: Different perspective on click (and all synchronous handlers) - by the time you step over to the next line, the listener has executed, we need the listener to run in the provenance of the surrounding context - any other default will lead the user to incorrect state
            - think of specific use cases, e.g. performance tracing - if you're trying to figure out how long something takes, then if click() is called programmatically, then we need to account for work in that handler
            - other example is DI, context is a global service map; click having access to global/scoped service map at registration time seems incorrect because code invoking event listener could be coming from different place; contract w/ click handler extends to any code invoking it

    - Dan: Is programmatic click() in context of your own delegation framework?
    - Jatin: could be custom HTML events delivered synchronously
    - Dan: Normal case for click() is coming from HTML, how to handle this?
    - Jatin: empty
    - Dan: So how does DI work in the empty context?
    - Jatin: need event delegation, doesn't work so well without it, that's another issue, I think it needs to be built-in; custom events never triggered by browser
    - Andreu: custom element inside custom editable, deleting element triggers callback from outside JS
    - Dan: register event handler from inside DI context, so registration context makes sense to capture there? but is that relevant for frameworks?
    - Jatin: every framework has a bit of delegation, even classic frameworks
    - Dan: or maybe it actually needs to be based on widget tree
    - Jatin: DI use case is maybe less clear
    - Dan: tracing is also confusing - say you fetch and wait for return and then do something after, we lose the intermediate work
    - Steve: our tracing is fine with that, spans are mutable, span tree reflects the call tree
    - Dan: so how does this affect events?
    - Jatin: API to either create a root trace if one doesn't yet exist, or else (if it already exists because invoked synchronously) then make a child span
    - Dan: gotcha, so that's why empty makes sense, don't want to inherit the wrong root trace from parent
    - Jatin: for DI, service map available to all contexts, singleton needs to be singleton everywhere - e.g. singleton map would exist outside the AC; scoped maps would be more registration time; could work w/ dispatch-time, but scoped maps may not always have what you want - this may be the intention, e.g. react contexts don't guarantee you're going to find what you're looking for
    - Steve: lots of contexts - dispatch, registration, construction, etc
    - Jatin: collection of many event handlers, each could have their own context. Another super-interesting context is load-time context - for dynamic loaders, may want to know when a class was loaded
- Dan: don't know whether frameworks want to use snapshots this way - do we need to make snapshots more efficiently than making a whole new object? Would be messy, but could allow "adding" a snapshot to an existing object.

- Jatin: also haven't discussed server-side - this is also available on servers, right?
    - Andreu: yes, web integration is a requirement for 2.7, but we hope server integration would be built on consistent principles
    - Jatin: AsyncLocalStorage exists, right?
    - Dan: yes, but it lets a lot flow through with causal context; we should file an issue w/ NodeJS's standards positions repo - also w/ cloudflare and deno
    - Andreu: discussed w/ Guy Bedford
    - Dan: WinterCG wasn't sufficient, need more Node feedback; hope for consistency between web and node, but they might go w/ dispatch regardless of what web does
    - Dan: Andreu was working on enumerating, but most events didn't *always* have a dispatch context, and feedback to avoid switching between dispatch and registration led to always going with registration
    - Andreu: Some events are always sync, but we need to have a single API for every event
    - Jatin: one thing that's super clear - any attempt to make context propagation easy will end up causing big problems for frameworks; example - thread pool on servers, grab a thread to process a request and then return it; problem arose when wanting background thread to have access to request thread - now threads being reused and risks leaking data across users; not trivial to fix, requires exposing gory details. Less likely on web, but multilogin is still a thing and we don't want user A to pollute user B (account-switching menu at the very least is dangerous territory)
    - Steve: is the tl;dr that we need to make sure that the gory details are at least accessible?
    - Jatin: yes, the consequences can be really bad, if you want a reasonable default maybe don't use this API.
    - Dan: but at the same time, we absolutely need a reasonable default. Tracing can't require calling obscure APIs.
    - Peter: it becomes global - not just people intentionally using AC, but everyone who calls a function needs to be careful.
    - Steve: best we can do is have people in dangerous territory pay attention and then do the right thing for ordinary cases
    - Dan: I thought registration was the best fallback because of object capabilities - principle is that trusted code is running in a particular context and you don't want it to be able to escape that context - code has certain rights; if the outer (more powerful) code wants to set the user to a particular value and invoke inner code, the inner code can never reset the value of that variable, all sub-tasks

calling back into user APIs are guaranteed to be restricted to that user - this is an argument for registration over empty
    - Dan: Could register click handlers in a no-span context
    - Jatin: Event delegation …
    - Dan: We care more about what information is exposed to framework
    - Peter: On the Zalgo question - different behavior is registered callback triggered by code or user interaction - this doesn't seem like a bad Zalgo situation - it's generally distinguishable which case you're in; if JS is invoking some code then it really does make sense to propagate context through. If you don't allow the context to flow through then it's very difficult for user code to handle correctly - this is particularly bad for custom elements, but applies elsewhere
    - Steve: Sorry for leading us astray with registration; next step is to enumerate events with Andreu?
    - Andreu: concerned about async events - will try to get fuller list.

Jul 23, 2024

- Short topics
    - Editorial: unify term async context mapping #99
        - Dan: suggest use term “snapshot”
        - Shay (chat): I like "snapshot mapping" or "snapshot map" here because it sounds like you're going to have to talk about "snapshot entries" as well
        - Shay (chat): (and then the internal thing is the "variable key"?)
    - Web integration document: https://github.com/tc39/proposal-async-context/pull/100
        - Andreu: not all events are shipped initially with snapshot captured
        - Andreu: Dan Minor want the document be a requirement for stage 2.7
        - Dan: focus on cases where event dispatched with snapshot captured, not expanding on spectacular cases
        - Dan: Yoav Weiss mentioned in task attribution that capturing snapshots in WebIDL might be too slow
    - AsyncContext Update July 2024
        - Dan: additional slide about AsyncContext & web frameworks
            - If frameworks take snapshot objects all the time, will it be too costly to have identical objects with the same snapshot?
        - Shay: good to be cheap to take snapshot
        - Justin: it is slow in react context with recursive reconciliation.
        - Shay (chat): I keep wanting to hope that this would let you legally read from context in more places/do it more lazily but I'm not sure that works out for react the way it it does for fine-grained frameworks
        - Shay (chat): keep running into "oh, but it'd just end up rerunning the component anyway, no savings there"

    - Performance concerns about frameworks making lots of new snapshots
        - We could alleviate this by having Snapshot.get() that returns the same object to avoid allocations
            - would need to be frozen to satisfy SES requirements
            - would allow observing whether someone else mutated a variable (side question: does a no-change mutation create a new snapshot? Shay Lewis hopes so)
            - would be unprecedented w.r.t. the spec creating frozen objects
            - also some cross-realm concerns
        - Plan is to mention that we're investigating framework use and performance, and to lightly allude to there being some trade-offs, but not go into detail about the nature of the solution and thus avoid rat-holing on those issues.
    - Viability of mutable variable over set/enterWith
        - Chengzhong Wu (legendecas) works on a document to cover this part.

Jul 9, 2024

- Short topics
    - https://github.com/tc39/proposal-async-context/pull/97
    - https://github.com/tc39/proposal-async-context/pull/96
    - https://github.com/tc39/proposal-async-context/pull/94
- Longer discussions
    - Cross-realm variables (not a blocker)
        - In HTML EnvironmentSettingsObject used to determine the incumbent realm could be built on AsyncContext
        - Might be relevant to https://github.com/tc39/ecma262/issues/3160
    - Matteo: variable is only accessible in .run function, have to wrap user code in `.run`. Not a problem for making frameworks, a problem for dev modifying their entry points for default value.
    - Andreu: there is an option bag for default value, `new Variable({ defaultValue: 123 })`
    - Steve: that is one value for all default cases
    - Matteo: pre-require a module has no chance for .run
    - Andreu: if you have user code in the top level of multiple modules, I can see how having to wrap all of them would mean adding boilerplate
    - Stephen: .set can take in a place that make sense for a scope.
    - Steve: user would ask for .set in both variables
    - Stephen:
        - example: create a DB in one file, access it in another - forward-flowing (flow-thru) allows this use case
        - .run and .set both make some degree sense with flow-thru
    - Matteo: https://gist.github.com/mcollina/825638bedf35c49411733243c44acb4e

        - use case: beforeEach() in a test can't initialize an async var that requires a run API
    - Steve: can we approximate this by explicitly running with a "scope" rather than a "value"?
        - ```
        - async_context_scope(() =>
        - variable.set() // only effective in this scope.
        - })
        - ```
    - Stephen: force scope heavily limits the constructs.
    - Andreu: web api integration details can depend on strict scoping (for async web APIs that take an async callback, such as `document.startViewTransition()`), and change to the .run model could delay the web integration. Not saying this shouldn't do.
    - Steve: my stance is that I'm hoping we can avoid delaying past August for attempting 2.7
    - [ ] A summary of .set semantics is needed to move forward with this semantics.

Jun 25, 2024

- Short topics
    - https://github.com/tc39/proposal-async-context/issues/90 context for the `error` event – how do we make that happen spec-wise?
        - Andreu: Domenic talked about line and column numbers attached to the error event. Ecma262 don't have any infra about capturing contexts at throw completion etc.
        - It would not be a reasonable timeframe to get this into the HTML spec/ecma262 before the next plenary
        - Steve: are there any use cases for the contexts of the error event? Are they related to the line/col numbers?
        - Andreu: ecma262 don't have any line/col numbers in the spec texts.
        - Steve: the proposal should be caring about if restoring the context or not
        - Andreu: in an eventual consistent future, it would be specified in the same place
        - Steve: what is the use case for restoring the throw context in the error event handler. Divergence with the registration-time of the event handler. If we ultimately follow-up with a separate causal-context type of variable, would all the use cases here just rather use that other kind of var?
        - Andreu: If unhandledrejection restores the rejection context then we want error event to restore the throw context for consistency; Bloomberg has a use case for the former but not the latter

        - Chengzhong: Can ErrorConstructor capture the context as a property like `.stack`? Implementation like V8 has infra capture stack at throw time but it didn't exist in ecm262
        - Andreu: We discussed about Promise constructor-vs-resolution-vs-schedule time, need to investigate the use case of Error instances.
        - Steve: if in a follow-up proposal of causal variable, the follow-up proposal might be different from the current one. It's not clear that two separate variables will be sufficient.
        - Andreu: Bloomberg has a use case about the unhandledrejection contexts. Whenever an application throws, kill that specific application (via AbortSignal in context) and leave other applications to continue to run. A global unhandledrejection handler, each application gets its own signal when it starts, but inside the application code they would not care about flow-through vs flow-around
        - Steve: this is an example where there's some mixing between (registration|lexical|flow-around) and (causal|flow-through), since AbortSignal tends to want to behave lexically, but here it wants causal context in rejection handler
- Longer discussions
    - https://github.com/tc39/proposal-async-context/pull/94 documents continuation flow to be a follow-up proposal
        - Chengzhong: there's a more nuanced distinction - it's not that each variable follows a flow-through or flow-around semantic; restoring the most recent causation allows to observe previously-unobservable promise state.
        - Steve: two types of flow: lexical vs causal; at least three different ways to decide which wins when contexts merge: (1) determined by the type of merge, (2) choose separately for each variable, (3) manual override when registering a callback (i.e. `wrap`). This is a two-dimensional matrix (variables on one axis, merge type on the other) and it's not clearly separable into a single dimension
        - Daniel: in Bloomberg's case the two variables both work, and passing the causal context as a variable would also work.
        - Andreu: To have the registration time for events, and an AsyncContext.Snapshot property on the event for the causal context
        - Steve: worried that if we have two type of variable,
        - DE: existence of merge points, not convinced that purely causal make sense
        - Steve: usercode needs to be ignorant of async context variables. The variables need to decide which wins at merge points
        - DE: Given ALS solves problems even though it might not solve it perfectly. This big problem space doesn't mean we shouldn't do anything

- Andreu: Many of us don't have the full picture, can we get feedback from developers that use ALS and other solutions in other languages, as well as from APM vendors?
- Chengzhong: Alibaba's internal APM solution is based on OpenTelemetry, which uses ALS. Talking with other APM maintainers, ALS matches other languages in general, in alignment with OTel Spec. The semantics of web APIs might not be fully useful in detailed specific semantics. OTel doesn't specify anything language-specific, just a general approach.
- Dan: Do other languages point to a consistent answer for async/await? Is it documented?
- Chengzhong: My PR document is for promise construction and promise APIs, how it matches the current model.
- Steve: Registration-time semantics is the clearest possible semantics, that seems like the right default. It's clear we need some mechanism to expose the other flow, but the default should be registration-time, so it's reasonable to move forward with this.
- Chengzhong: We need to document the current semantics we're choosing. We can continue iterating on that document for any semantics with the two variables. We should expand on any ambiguity on how the two semantics might be different in some cases.
- Steve: So `error` and `unhandledrejection` would be registration-time, but the causal context would be provided some other way?
- Dan: That's what I'm leaning towards.
- Steve: We would then not have a separate variable type, and expose the causal context whenever needed?
- Dan: It's easier for me to imagine exposing it for events than promises.
- Steve: If you're exposing this other Snapshot, do we need a more performant way to get back into that snapshot? Stephen's complaint was that wrapping the code with `run` would kill performance.
- Dan: That request is broader than that, since it relates to `enterWith` and multiple variables in one shot. I don't know what's actionable, since `enterWith` is unworkable (stack discipline), and multiple variables would still need a callback. If someone thinks callbacks are slow, they need to provide a benchmark.
- Steve: Do we need a whole chain of causation, since a single snapshot might not give you what you need?
- Dan: You could build chains of causation in a variable with nested `.run`s, and that's what task attribution does. It has a high runtime cost.
- Steve: This is why you'd want to have each merge type have its own place where it stashes its alternate context.
- Dan: Certain particular merges would give you access to the causal context. We'd need a convention so it's easier to use.

- Issue Triage
    - https://github.com/tc39/proposal-async-context/issues/93
        - Andreu: import defer proposal? How this works with ShadowRealms? Context variables won't be able to cross the boundary, only functions, but a ShadowRealm run can call into the parent realm, and the context should flow through that.
        - DE: Everything should drive from where ShadowRealm should be instantiated (??) to avoid empty root context or zalgo.
        - DE: Web platform needs to decide what needs to be agent-wide variable and realm-wide variable to isolate access.

May 28, 2024

- Attendees: Steve Hicks, Stephen Belanger, Daniel Ehrenberg, Chengzhong Wu, Matteo Collina, Andreu Botella, Justin Ridgewell
- Longer discussions
    - Error events https://github.com/tc39/proposal-async-context/issues/90
        - Proposed solution: capture the context at "throw"
        - Dan: Seems like these semantics are best, if they're implementable.
        - Andreu: I have an idea for how to implement this, but I'm not sure
        - Steve H: Is this still the same question independent of whether we flow around or through?
        - Probably yes, it's the same question
        - Chengzhong: Dan mentioned in the chat that context can be captured as a field of the Error object like stacktrace
        - Andreu: Maybe, I'm not sure. Relates to unspecified stack handling. I'm not discarding it.
    - Stack of async/await functions
        - Stephen B: For the separate case of stacks of async functions: Synchronous code which throws is automatically raised to the caller, and async functions are not. I think of the await as handling. It's explicitly handling, "I'm now responsible for this" if an error happens. The point where unhandled happens is where the outermost await happens.
        - Andreu: I think both cases have to be considered together, because it's the same error handling mechanism, one for sync and one for async.
        - Stephen B: Unhandled means there's nothing to propagate to. When you throw, it will walk up the sync call chain, and then call uncaught exception. Mechanically, async/await does something at each await
        - Dan: That corresponds to the implementation, but I'm not sure if it's programmers' mental model
        - Andreu: Flow through vs flow around is related to this
        - Matteo: What I recommend is to minimize surprise behavior for end users between sync and async code. In the eye of most JS developers, there is

no difference between a function and an async function. People use try/catch for both. We need to look at both, and their correspondence.

- Next steps on the extension with the flow-through model
    - Stephen B: Flow-through minimizes surprising behavior. Customers are confused because callbacks flow context into the logically continuing code, but with async/await, the logically continuing code is what happens after the await. Different mechanically, but from the user code perspective, what they're expecting it to work the same way whether it's callbacks or promises.
    - Steve H: I wonder if this is transient, just when trying to transition. When async-first, maybe it's surprising the other way. An async function feels like a separate task, so the follow-on is the previous context.
    - Chengzhong: I'm not clear on some details of the flow-through model, given the lack of spec text/detailed semantics. In Stephen's doc, it only defined the higher level model. I'd like to have detailed spec text, so that we can compare the semantic differences.
    - Stephen B: I could possibly take this on when I'm back in 2 weeks. To Steve H's point: I don't think this is the case. Flow-around makes sense with UI libraries where it's working recursively and it doesn't flow backwards in a logical form. In Node, we have all this branching and converging and we want to know what came before me. We need to be able to follow, where did we follow-from. We're supposed to create follow-from links for all of the preceding things for Promise.all, I care less about that, right now we have to guess where we're linking to. Guessing is bad.
    - Matteo: The difference in pattern between browsers and Node.js: in the browser, when we talk about async/await and Promises, there are fewer Promises because more Promises cause more lag on the UI. In backends, you're doing a lot more async work, up to thousands of promises. If you call an async function, the context stays there. I think it will require way more effort for developers, from APMs and framework authors, to give us something that's consistent with user expectations. Certain cases might not be fixable to some extent. Startup and metaframeworks: Next.js's request store is handled within the metaframework. Without a metaframework, there are way too many quirks, so a developer cannot handle it. So it's nicer to have enterWith, so you don't need to wrap everything. Is this too costly, performance-wise? This goes to our friends in engines to decide. We need to have a measure of what the problem.
    - Dan: the current AsyncLocalStorage is not flow-through matches the AsyncContext proposal. This is the current reality. How bad the current reality is with concrete example. My current understanding is that both context matters. And Promise.all is one of the case that both concept

come together. The current proposal chooses a subset that seemed good enough.

- Stephen Belanger: APM in all other languages cost less than 5% overhead. We have to guess the order of awaits.
- Steve H: in a properly instrumented code, correct graph can be built
- Stephen B: Can not instrument all the userland code
- Justin: There was an example in Matrix for this case https://matrixlogs.bakkot.com/TC39_Async_Context/2024-05-21#L23
- Dan: can you come up with a python reference example or any OpenSource reference that works as described.
- Stephen B: Other languages can do bytecode manipulation but JS can not. Mostly internal APM vendor stuff
- Justin: .Net implements mutate and copies on start of an async function, it doesn't seem to do flow-through
- Dan: enterWith/set and flow-through/flow-around are different topics.
- Andreu: Even APM doesn’t care about merges, WebIntegration needs to merge contexts, like Observers like PerformanceObserver merges multiple contexts into a single event. The ecma262 has to be consistent with web spec and it becomes unclear about “consistent”?
- Stephen B: We only need one context to be a useful causal context
- Justin Ridgewell:.Net implements mutate and copies on start of an async function, it doesn't seem to do flow-through
- Stephen Belanger: We only need one context, it just needs to be a _correct_ context.
- Steve Hicks: if we're flowing through, the last promise to resolve is probably the most likely to have the most up-to-date data
- Andreu: {fill in}
- Justin: I disagree with every other languages behaves like what Stephen B described. No other languages operates like flow-through semantics. In there any languages implements flow-through semantics in the standard library?
- Stephen: No but every APM vendor has to implement that on their own. Ruby or Crystal might flow that way.
- Justin: .Net copies the current context value at the beginning of the async function with a mutable `.set`. Mutating with `.set` doesn’t modify the value in the parent scope.
- Dan: the rational needs to be really clear if we change the current proposal
- Steve Hicks: if we're flowing through, the last promise to resolve is probably the most likely to have the most up-to-date data
- Steve Hicks: copy-on-write seems like it would break "flow-out" semantics that Stephen wants?
- Justin Ridgewell: Yes

        - Stephen Belanger: Nope, it doesn't. You just need to capture at the end to flow out.
        - But it _does_ make that controllable.
        - Justin Ridgewell: As a return value?
        - Stephen Belanger: Essentially, yes.
        - But doesn't need an explicit return.
        - Stephen: Just needs a hook to store and restore around the points.
        - Matteo: enterWith snapshot?
        - Stephen: I will be fine with landing some sort with flow-through spaces for future extension.
        - Dan: we should be explicit to the committee that there might be an extension with flow-through proposal
        - Stephen: We can not switch to AsyncContext in production before enterWith is with the proposal. Expecting enterWith to be bound with flow-through semantics.
        - Steve H: Is that a problem for SES folks to be about to change the value in outer scope?
        - Action items:
            - Write a spec text on the flow-through pattern to compare on async-sync code differences
            - Compare with other languages like Ruby with open-source APMs that adopt flow-through semantics
            - Give examples of the usefulness/essentialness of enterWith
            - Draft idea for snapshot enterWith and see if it fixes issues
            - Explain in some more detail what the cost of the current problems are (memory to process? Inaccuracy in guessing cause? others?)
- Short topics
    - Plans on next plenary
        - No update in the June plenary
        - Call on June 11 is canceled for June Plenary

May 14, 2024

- Short topics
    - `error` event on web integration and interaction with FinalizationRegistry (https://github.com/tc39/proposal-async-context/issues/82#issuecomment-2110595301)
- Longer discussions
    - The case for _not_ binding around awaits: https://github.com/tc39/proposal-async-context/issues/83
        - Steve Hicks: Are implementers happy about two way propagation?
        - Steven: Don’t have a strong opinion on the default contexts. There are different needs on the shape of the graph, needs a flexible way to retrieve

the necessary contexts, rather than dropping leaf contexts without a way to restore it.
    - Andreu: Will try implementing something, maybe `callingContext()`, in V8 and report on the complexity and performance difference.
    - Justin: Per-store propagation being configurable means O(n) restore operations. Snapshot/restore is a constant time operation. Would it be possible for engines to implement this behavior?
    - Stephen: Otel spec defines that links should be generated from Promise.all operations
    - Chengzhong: Otel spec is casually defined and mostly focused on links between distributed services. Not specifically mentioning Promise.all links
    - Stephen: diagnostic channel transform function can present to transform store data to span and may not present.
    - Stephen: Otel baggages set in different branches of an operation needs to be merged and propagated to subsequent sibling operations.
    - Stephen: No other vendors implement the proposed model. Want to create a new model.
    - Stephen: calling context can be a solution to only save span ids instead of span objects, in this way it would be cheaper to propagate span relationships. (??)
    - Steve Hicks: calling context may be feasible than per instance bind around await
    - Andreu: write down each own expected async/await and APIs shape and behavior.
    - Distinguished open questions:
        - Get calling context aside from the default context (like registration context)
        - Merging multiple context branches
        - Feedback to parent async context scope
- Per-instance wrap: https://github.com/tc39/proposal-async-context/issues/25
    - Stephen shared doc: Universal Context Management RFC and Universal Diagnostics Channel RFC, slides: Dynamically Flowing Observability Insight
    - Diagnostic channel only modifies one store

Apr 30, 2024

- Short topics:
    - `Variable.p.wrap`? https://github.com/tc39/proposal-async-context/issues/25
        - Worried about confusion with Snapshot.wrap
            - +2

        - Might want to snapshot multiple values
        - **Not convinced yet**
    - Specifying as weak references? https://github.com/tc39/proposal-async-context/issues/76
        - Weak values are slower
        - Implementers are already expecting it to be weak
        - Certain internal mappings that engines are worried will leak if they use AsyncContext, but they assumed normal Variables would be collected
        - **Informational note: programs could but probably won't GC, do not depend on it being implemented**, need to discuss in GH issue
- Longer topics:
    - Not binding around await (call-time resolve): https://github.com/tc39/proposal-async-context/issues/83
        - const { AsyncLocalStorage } = require("async_hooks");
        - const als = new AsyncLocalStorage();
        -
        - als.run(1, async () => {
        - console.log({ current: als.getStore() });
        - await new Promise((r) => als.run(2, r));
        - console.log({ current: als.getStore() });
        - });
        - Logs 1, 1
        - als.run(1, async () => {
        - console.log({ current: als.getStore() });
        - await als.run(2, () => Promise.resolve());
        - console.log({ current: als.getStore() });
        - });
        - Logs 1, 1
        - **Will discuss further with Stephen**
    - Calling context continuation: https://github.com/tc39/proposal-async-context/pull/77
        - APIS that might be merge points
            - Media event listeners might also represent a merge point, but not convinced yet
            - Signals
        - Promise.all/race and friend represent a merge point of multiple promises, might want to receive the inner promises' contexts after the await
            - Would also help Stephen's resolve-time need?
        - Re asyncVar.wrap(), maybe certain variables should behave differently

        - Could be set during construction, when awaiting they would receive different context times.
    - **Action Items**
        - event object property to receive calling context
        - don't wrap me `AsyncContext.dontWrap(fn)` api
        - Investigate receiving merged promise's contexts
        - AsyncVar construct-time option about registration or call-time
        -

Apr 16, 2024

- Hopefully quick topics (15min total):
    - Terminology: https://github.com/tc39/proposal-async-context/issues/73
        - Change to "mapping"
    - Receiver `this` during run: https://github.com/tc39/proposal-async-context/issues/80
        - Opposed, if pushback we will remove variadic args
    - Disposable Snapshot to support termination? https://github.com/tc39/proposal-async-context/issues/52
        - Proposal is to make Snapshot disposable, not add termination yet
        - Hesitant, because it's viral
        - Will discuss more on the thread, need to think though when to dispose `Snapshot.wrap()`
- Longer Topics
    - `using` syntax and Variable scope:
        - https://github.com/tc39/proposal-async-context/issues/60
        - https://github.com/tc39/proposal-using-enforcement/issues/1#issuecomment-2052638125
        - "Strict enforcement proposal" unlikely to help
        - Options are (1) lots of nesting, (2) one level of nesting with a runAll or runWithSetter, or (3) preferred, no nesting at all - but need to eliminate all risk of leaking mutations to callers, either by changing using or else some custom syntax (custom syntax would be ideal for adding this later)
        - There was a similar case in Rust wrt the destructor for a scope not necessarily running, in the pre-1.0 `thread::scoped` API (https://github.com/rust-lang/rust/issues/24292, a good write-up despite the name is https://faultlore.com/blah/everyone-poops/). Eventually replaced with a callback API (not unlike `.run`): https://doc.rust-lang.org/std/thread/fn.scope.html
        - AI: Justin to make a PR with specific needs for using
    - Call-time vs Registration-time context:

        - https://github.com/tc39/proposal-async-context/issues/18
        - https://github.com/tc39/proposal-async-context/issues/82
        - https://github.com/tc39/proposal-async-context/issues/77
        - Registration time is when you want to make the choice, but it's hard to "unwrap" at that point - could add a function exotic to do this
        - .NET has ExecutionContext.SuppressFlow and RestoreFlow
        - Some details around avoiding null mappings in DOM (question: are we sure that the null mapping is wrong?)
        - AI: explore unwrap method and addEventListener options

Apr 2, 2024

- Updates since last presentation
    - registration vs call time semantics
        - Justin will post calltime api
    - AysncContext.wrap is back
    - Andreu will post GH issue about which HTML APIs need to change
    - Small bug fixes that landed and refactorings
- What to do about Task Attribution's Call-time behavior
    - Generator's init-time vs call-time
        - Still need example from Stephen
    - RPC interceptor -> UI update
        - single registration handling multiple calls, want to propagate the the call time
    - signal setter -> reaction,
        - needs the call time context during some updates, but if multiple sets happen it's not clear
    - Async actions are simple, sync actions could be either.
        - Some events do not happen via API (click is via user interaction), and the empty context isn't useful there.
            - These are basically async actions
        - unhandledrejection is another async send, but the registration context is not useful
    - Task Attribution's `message` event?
        - Attribution uses call-time for MessagePort/MessageChannel, but not window.onmessage which uses registration
        - Task Attribution can patch Blink code to propagate a call-time context if necessary, can internally do a `snapshot.run(var.run(calltime))`
    - Zone.js allows patching the propagation to do call-time
        - Tell zone.js not to patch certain APIs, and those will carry call-time
    - Choices
        - If we do call-time
            - Possible to do registration-time via wrapping

        - But requires a lot of wrapping
    - If we do registration-time
        - Could do call-time via options-bag to every API
        - Not every API would require this (eg, `setTimeout()`)
        - Could be a boolean `AsyncContext.useCallTime` or better API
            - Maybe a wrapping function
            - Would need way to transition both ways
        - We could provide call-time context someway
            - `AsyncContext.callTime.run(() => {})`
                - Justin Ridgewell make an example using play on media element, similar to `signal.set()`

Mar 19, 2024

- Using Bindings
    - https://github.com/nodejs/node/pull/52065
    - https://github.com/tc39/proposal-async-context/issues/60
    - No to this feature because it allows polluting the context returned from a scope
    - Investigate as a follow on
    - Node shouldn't add this without AC support
- Generators (and by extension Iteartor Helpers) capturing context:
    - https://github.com/tc39/proposal-async-context/issues/75#issuecomment-1984151672
    - Reach out to @Qard to clarify the objection
    - Keep init-time spec for now, absent Stephen's use case requiring call-time.
    - If generators use init-time, then iterator helpers could use init-time as well.
- Stage 2.7: Next Steps
    - dminor asked for HTML integration spec before advancing to stage 2.7
    - https://github.com/whatwg/html/pull/9408
    - Collect Zone.js behaviors on Web APIs: e.g. the unhandled rejection behavior might not aligned with AsyncContext.
    - Dminor: listing APIs can also be helpful, but still would like to see exact spec.html. Happy to see an update on the next plenary and ask for stage advancement on June.

Mar 5, 2024

- https://github.com/tc39/proposal-async-context/pull/69
    - Registration time capturing is not possible when the observer callback is batched.
    - Registration time can be achieved by maintaining a WeakMap to the mutated element objects.

        - The PR will revert back to construction time.
        - https://github.com/tc39/ecma262/pull/3195
            - HTML currently has something like AC, but it's not properly implemented
- https://github.com/tc39/proposal-async-context/issues/75
    - @andreu is reaching out to champions to work with the spec
- V8 Design Doc: is close to finishing
- https://github.com/tc39/test262/pull/3874 Please take a look
- TODO: Collect what current polyfills like Zone.js do regarding HTML/Web APIs, as well as what task attribution needs. Once that is done, come up with guidelines for future web APIs.

Feb 20, 2024

- https://github.com/tc39/proposal-async-context/pull/68 Adds back AsyncContext.Snapshot.wrap
    - ecmarkup complaints about the spec language, fix ecmarkup or the spec language to land the PR.
- https://github.com/tc39/proposal-async-context/pull/69 Handle FinalizationRegistry
    - Registration time of event listeners is analogous to the construction time of FinalizationRegistry
    - This PR uses the FinalizationRegistry.register as the registration time instead.
    - FinalizationRegistry.register is not the time that a callback is provided, mismatches with event listeners.
    - This can be a correct precedent for future DOM APIs, like modification observers.
- https://github.com/tc39/proposal-async-context/pull/70 Only preserve AC for user generators
    - Iterator helpers can be a follow-up.
- https://github.com/tc39/proposal-async-context/pull/71 Update polyfill
- Using a linked list in the implementation instead of a map
    - It is really difficult to flat a linked list, which is important to free the unobservable list nodes and preserve the tailing one.
    - Must wait until AC stack is empty to due a mutative flatten: https://gist.github.com/jridgewell/ef8a674291f8f7419a2bea0448c3b0eb
    - Possible to do immutable, but incurs memory after flattening
- Feedback from V8 re. data structures
    - Alternative: Hash Array Mapped Trie (HAMT) can provide better memory usage
    - V8 will not land anything before the design doc is approved, and also not until stage 2.7
    - When setting same value again, only reallocate on a single branch of modification. linked list reallocates the whole tree to keep immutability
- Benchmark suites: adding cases integrated with OpenTelemetry, with Zone.js and with AsyncContext

Jan 23, 2024

- FYI: Working on benchmark suites https://legendecas.github.io/Speedometer-asynccontext
    - Add async/await transformation case @legendecas to improve the necessity of CPED
- Justin's summary:
    - Not clear the extent of APIs that need to support snapshot/restore, would be nice to have a (probably non-exhaustive) list:
        - setTimeout, addEventListener, …
        - Any in TC39 side? I don't think so, only Promise APIs
    - Add wrap method (static, maybe both static and prototype)
        - Two types of wrap, static and prototype wrap
        - Just do static wrap at the moment
    - Expand prior art section, explaining the difference between the current AC proposal
    - Section detailing how to handle libraries that don't support and aren't updated to support propagation
    - Section detailing alternatives:
        - Every closure captures a snapshot (ie, like Generator now does, but everything)
        - await hook so userland could implement instead of directly in language
            - This would introduce security concerns
            - The explainer should stress the performance instead
- https://github.com/tc39/proposal-async-context/pull/41
    - The current implementation didn't capture correctly the context during the rejection
- https://github.com/nodejs/node/pull/48528
    - @abotella mentioned that the blocking issue should have gone
- Embedder API
    - Focus on performance rather than API nicity
- HTML Integration
    - @abotella working on rebasing task attribution to work with AsyncContext. Currently task attribution only works on the main thread, but scheduler.yield must work in workers as well; the refactor uses a single API that can propagate the AsyncContext snapshot, task attribution (where applicable) and scheduler.yield stuff in every thread.
    - Prioritize deciding the HTML semantics over making task attribution work
    - Triggering the continuation of TPAC discussion on the propagation semantics of events.
- Built-in iterators and iterator helpers integration
    - The spec defines both as generators, and currently they preserve the snapshot.

        - For Map/Set built-in iterators, this is not observable.
        - For Array built-in iterators, this can be observed if the array indices have getters, since those are called with that snapshot.
        - Also observable when the iterator helper methods are called.
    - It probably doesn't make sense that Array iterators preserve the snapshot only for getters.
    - Resolved that spec-created generators will not necessarily preserve a snapshot, but JS-created generators will.
    - Open an issue to discuss the difference between spec-created and JS-created generators, and a PR to make this spec change

Jan 9, 2024

Agenda Items:

- https://github.com/tc39/proposal-async-context/issues/65
    - Web integration is a hard requirement
    - Web compatibility on adding or removing async context support for a particular Web API should be deferred to WHATWG/W3C.
- V8's plan to get rid of CPED
    - Need a web benchmark to convince V8 about the performance cost
- Justin can invite Google APM (frontend) framework team to the call next time

Dec 12, 2023

Agenda Items:

- Mozilla may try the proposal in Jan
- Current WIP V8 C++ API can be improved on the Snapshot for TaskAttribution's use case: take a snapshot and set a variable at the same time.
- V8 wants to get rid of CPED because they're concerned about the performance impact of passing the embedder data through promise jobs. This affects AsyncContext because it would also have to pass the snapshot pointers through those promise jobs. Andreu did a synthetic benchmark measuring only promise continuations, and those fields do make a difference, but they haven't done realistic benchmarks.
    - One suggestion was to use Node.js's implementation of fetch as a benchmark, since that uses promises heavily.
- Andreu is working on refactoring Chromium's task attribution to use AsyncContext. They tried making every event use the registration-time snapshot (which involves cloning the snapshot map at every event), and ran Speedometer on that. There was a regression (compared to CPED-based task attribution), but it was slight and the ranges overlapped significantly. But that might not reflect cases with multiple AsyncContext variables.

Nov 14, 2023

Agenda Items:

- https://github.com/tc39/proposal-async-context/pull/63

The calendar item didn't suffer from the daylight-saving time change. We may need to re-arrange the call time if people have conflicts with the new meeting time. Please follow the conversation in the matrix channel if you would prefer a new meeting time.

Oct 31, 2023

Agenda Items:

- No updates this week: Chengzhong Wu (legendecas) is working on a doc about incumbent settings and active scripts in HTML, and the status of browsers implementations to see if AsyncContext could take place in.
    - Prioritizing design doc.
- Andreu: V8 landed a change on CPED but got reverted for wpt regression of task attribution
- Andreu: List of event semantics (reg vs call)
- Yulia's confusion: what AC.Snapshot is for, explaining why it is essential for span-id propagation
- FAQ: why does run takes a callback, not a enterWith method. Chengzhong Wu (legendecas)
- Dan and Andreu are going to present at NodeConfEU: https://www.nodeconf.eu/daniel-ehrenberg-asynccontext-observability-in-javascript-through-tc39-and-wintercg

Oct 18, 2023

Agenda Items:

- https://github.com/tc39/ecma262/pull/3195
    - It would be good to unify the problem space around incumbent realm and active scriptOrModule under async context. Chengzhong Wu (legendecas)

Sep 19, 2023

Agenda Items:

- Quick update: slides for Sept 2023: Async Context 2023.09
- TPAC updates?

Sep 5, 2023

Agenda Items:

- Slides for Sept 2023: Async Context 2023.09
    - Elaboration on unhandledrejection: Deferred Promise Jobs & AsyncContext
- Problems in call-time promise unhandledrejection context with Node.js async_hooks
    - https://github.com/legendecas/node/tree/async_hooks/call-time

Aug 22, 2023

Agenda Items:

- Items needs to be addressed before the 2023.9 TC39 meeting:
    - Still have bug in unhandled rejection, pending PR https://github.com/tc39/proposal-async-context/pull/41
    - Need to verify spec behavior propagates context at time of `reject()` call: https://github.com/tc39/proposal-async-context/issues/16
- engine262 implementation: https://github.com/engine262/engine262/pull/227

Action Items

- [ ] chengzhong.wcz@alibaba-inc.com to work on implementing reject call-time in Node via flag

Aug 8, 2023

Agenda Items:

1. https://github.com/tc39/proposal-async-context/pull/61
    a. Merged
2. Andreu has engine262 implementation, no PR yet
3. Test262 tests: https://github.com/tc39/test262/pull/3874
4. Still have bug in unhandled rejection, pending PR https://github.com/tc39/proposal-async-context/pull/41
5. Need to verify spec behavior propagates context at time of `reject()` call

Async Context Meeting (2023-08-08 13:02 GMT-4) - Transcript

Jul 25, 2023

Agenda Items:

1. https://github.com/tc39/proposal-async-context/issues/18#issuecomment-1645671319

    a. Current expected behavior is fine. See if V8 implementation is not complicated or not slower, otherwise we can revisit.
    b. AP: Prepare examples for iterator helpers to see how the context is propagated.
    c. Are there any built-in iterators not built from generators which observably don't propagate the context? AP: Find if there are and discuss them.
    d. Confirm fast-paths for array/string iterator still works with context restoring
2. https://github.com/whatwg/html/pull/9408
3. Task Termination

Async Context Meeting (2023-07-25 13:02 GMT-4) - Transcript

Jun 27, 2023

Agenda Items:

1. Node.js semantic differences discussion
    a. Thenable handling
        i. Bug in V8 that effects Task Attribution, Cloudflare local storage, and new AsyncContext code
    b. Unhandledrejection
        i. Node has decided to use `reject()` time context, because it's what embedder preserved continuation data gives us
            1. Deferred = ctx.run(1, () => {…})
            2. ctx.run(2, () => {deferred.reject(1)})
            3. Would inherit 2 at unhandledrejection
        ii. Would be nice if we could have both init and call time contexts at unhandledrejection, need to pass snapshot to handler to happen
    c. Async generator
        i. Committee prefers init time capture and restore
        ii. DataDog is fine with that approach
        iii. Not clear if Node has the tools to implement it
    d. Per agent vs per realm context sharing
        i. Current spec is per agent, so different realms can capture the context
        ii. Node chose to make it per realm
        iii. Node attaches symbol to async resource, eg the literal promise instance, so data can leak
        iv. Might be fine to use current spec since there's not the same leakage unless you have the actual asyncVar instance
    e. https://github.com/nodejs/node/pull/48528
        i. Current impl uses a linked list of frames, run pushes a new frame to the list

        ii. Want the ability to snapshot only a single var's value to be restored later on top of whatever is currently present
        iii. Const fn = storage.bind(() => {})
        iv. fn()
2. Updating Design Doc
    a. V8 had concerns about current impl
    b. Must faster than async_hooks for node's benchmarks
    c.

Jun 13, 2023

Agenda Items:

1. https://github.com/tc39/proposal-async-context/pull/55
    a. Feedback from web engines hackfest
        i. Snapshot.p.restore might be confusing, thought it restored the context permanently
        ii. Several attendees expressed concern
        iii. No objection to switching back to `run`
        iv. Will switch back to `AsyncSnapshot.p.run`
2. Heard feedback from Chrome and Webkit
    a. Want to see speedometer benchmarks for performance, and worried about memory usage
    b. Need to include in design doc how there will be no regressions
        i. If task attribution is implemented based on async context
        ii. Task attribution and AsyncContext could coexist in a tuple structure, so that attribution does not need to use the context mapping for lookups/set
        iii. Schedule priority will already be storing a second bit in the continuation data, so adding another pointer to a mapping context might work?
    c. Need more explicit data structure guidance
3. Task Termination discussion
    a. Need some form of reference counting for a fast path
    b. Snapshots would allow references to continue tied to the snapshot
4. WIP PR to HTML spec
    a. Prior task attribution PR was reverted because of performance issue
        i. It added context to all possible callbacks
        ii. https://chromium-review.googlesource.com/c/chromium/src/+/3738432
    b. Currently wrapping every callback
    c. Event related callbacks are performance sensitive
    d. If the benchmark doesn't use AsyncContext, hopefully no regression
    e. If the code does use AsyncConext, how much of a penalty are we willing to take?
5. Angular
    a. Want to use AsyncContext in new Signals implementation to support async computation (not just old Zone.js code!)

Action Items

- [ ] Review reverted Chrome PR Justin Ridgewell
- [ ] Add more details to design doc
- [x] ~~Rename `Snapshot.p.restore` back to `Snapshot.p.run`~~ ~~legendecas@gmail.com~~

May 30, 2023

Agenda Items:

1. Actions from the last plenary meeting TC39 Meeting Notes — 2023-05-17
2. https://github.com/tc39/proposal-async-context/pull/55
    a. Namespace
    b. Name of AsyncLocal

Attendees: abotella@igalia.com chengzhong.wcz@alibaba-inc.com dehrenberg@bloomberg.net Justin Ridgewell

Notes:

- V8 work:
    - https://chromium-review.googlesource.com/c/v8/v8/+/4553118
    - https://chromium-review.googlesource.com/c/v8/v8/+/4475425
    - https://docs.google.com/document/d/1UewS3CiSbghRuOjSDOmH5FAPeDQ94ER4QKJPlgnCLK8/edit#heading=h.mwad14vicl1e
- Async Context Meeting (2023-05-30 13:07 GMT-4) - Transcript

Actions:

- [ ] Longform examples that aren't trivial enough to fit in slides
    - [ ] Should also show how multiple instances will be used together
- [ ] Finish design doc and V8 implementation to get actionable feedback from V8 delegates
    - [ ] Including which data structures to use for performance
- [x] ~~Renames~~ ~~legendecas@gmail.com~~
    - [ ] AsyncContext namespace
        - [ ] Variable class
            - [ ] Get
            - [ ] Run
            - [ ] Get name
        - [ ] Snapshot class

        - [ ] Restore
- [ ] Still meet with Node Diagnostics WG
    - [ ] legendecas@gmail.com Justin Ridgewell
    - [ ] Feedback on unhandled rejection context
    - [ ] Feedback on generator/async-generators

May 2, 2023

Agenda Items:

1. Task termination
2. Kevin's feedback on generators: https://matrixlogs.bakkot.com/TC39_Delegates/2023-03-23#L173
3. Open issues
    a. https://github.com/tc39/proposal-async-context/issues/49
    b. https://github.com/tc39/proposal-async-context/issues/50

Attendees: abotella@igalia.com chengzhong.wcz@alibaba-inc.com dehrenberg@bloomberg.net Justin Ridgewell tc39@softwarechris.com yoavweiss@google.com

Notes:

- Async Context Meeting (2023-05-02 13:10 GMT-4) - Transcript

Actions:

- [ ] Spec on snapshot/restore with generators
- [x] ~~Constructor extension: name and defaultValue in an option bag~~
- [x] ~~Splitting into two classes: AsyncContext and AsyncLocal (name pending)~~
- [ ] Unhandledrejection reachout to Node.js Diagnostic WG

Apr 18, 2023 | Async Context Meeting

Attendees: abotella@igalia.com chengzhong.wcz@alibaba-inc.com dehrenberg@bloomberg.net Justin Ridgewell ljharb@gmail.com tc39@softwarechris.com yoavweiss@google.com Mathieu Hofman

Agenda items:

1. Cloudflare's practice @jasnell

2. Constructor Extensions #29
3. Ergonomics of HOFs #21
4. Generators #18
5. unhandledRejection #16
6. Name bikeshed #44

Notes

- Chengzhong is working on a WIP design doc and implementation in V8: V8 AsyncContext Design Doc. The design doc talks about performance and embedder requirements. Not yet ready, Andreu will be working on that as well. It might need changes to how Chromium implements task attribution.
- Chromium's task attribution handling currently has a bounded list of tasks that are tracked, and old tasks are dropped when the bound is passed. For Blink tasks, this can be tracked, but for V8 tasks in AsyncContext this would need some refcounting or some tracking in V8. Yoav tried to add code in V8 to let the host know about task termination, but it's in a hot code path and V8 folks are reluctant. Yoav is composing a doc about this.
- Constructor Extensions:
    - Justin: The default value only happens on non-existing value.
- HOF:
    - Dan: prefer a class with a run method, like AsyncResource.
    - Mathieus: concern on losing the ability to create a function to snapshot the async context
    - Dan: wrap can be a helper method on the bind

Action items

- [ ] Reach out to React folks to talk about use cases, like async components
- [ ] Clarify the use cases in the readme. This is the limiting factor for further advancement of this proposal.
- [x] ~~Kevin's feedback on generators: https://matrixlogs.bakkot.com/TC39_Delegates/2023-03-23#L173~~