JS Numerics community call

2026-03-26

Attendees

Agenda

- Sequence unit formatting
- Fallback logic for existing, non-protocol implementing, object parameters
- Drop rounding mode options

Notes

Drop rounding mode options

EAO: What are the use cases for rounding modes other than round-ties-to-even?

SFC: If you start with a Number amount and do a conversion, you end up with a decimal amount. Going from binary to decimal is where a user who wants to trunc should be able to trunc. No need for use cases when formatting number. Trunc is common. No less motivated than general rounding mode use case.

JMN: We settled on using Number as basis for conversion. Number is the data model, we support strings but ultimately we work with Numbers.

EAO: Not removing rounding for formatting, but when constructing Amount / converting between units. Still think examples are needed

SFC: You want to display your height and you want to not exaggerate, or you want to always exaggerate. You have to set rounding mode.

EAO: Plausible. Link to rounding mode proposal?

JMN: how long have different rounding modes been supported?

SFC: forever in intl number format, referencing numberformat v2/v3 which added additional. Trunc, floor, ceil, half-expand always there, others added in v2/v3

RGN: Number defaults to half-even, NumberFormat defaults to half-expand

EAO: data useful for non-default rounding modes?

SFC: surprising to have rounding modes work in Temporal.Duration but not Amount. If any mode motivated, all modes motivated

EAO: address feet/centimeters issue on PR? I agree that given that we have rounding mode as pre-existing thing with particular list of options, we should support that list

JMN: In Decimal context when talking about rounding modes we took inspiration from numberformat and temporal, some pushback from WH and others saying some of these things just aren't implemented at C++ level. Stick to IEEE 754. Makes sense for Decimal, harmless in our contexts because we don't do arithmetic?

SFC: We're not doing IEEE rounding.

Sequence unit formatting #398

SFC: Recent comments on issue.

EAO: When we look like what we're supporting, DurationFormat does close to unit formatting for d
Duration units, if we build an interface that looks different from DurationFormat, we should have a good reason.

SFC: the central question: are sequence units a scalar indexed in some unit of measurement (largest unit, smallest unit), or expressed as a bag of scalars? One scalar per unit in sequence unit. Issue with using scalar: all the design questions apply here. We have value field that's either a string, BigInt, or Number, and that applies. If we say sequence units a bag of scalars, several ways we can express that. Value field becomes an object with fields.

SFC: Option D seems cleaner than A:

```
// A
numberFormat.format({
   value: { foot: 3, inch: 7.8 },
   unit: "foot-and-inch"
})

// D
```

```
numberFormat.format({
   foot: 3,
   inch: 7.8,
})
```

SFC: I feel that sequence units should be treated like regular units. Give them a scalar, sequence should be formatting construction. Otherwise treat as regular unit. Other thoughts? EAO point that DurationFormat uses bag of options should be discussed

EAO: cleanest: supporting object w/ foot and inch as formattable thing supported by NumberFormat, allow it to show up in value field. Value that's an object that's foot and inch. Also explicit unit required. Unit is foot and inch, we expect object w/ value field that's object with foot and inch. Units.xml contains single-unit conversions. Avoids issues of whether with single scalar value is measuring feet and inches.

JMN: what about original version?

SFC: was hoping we could punt no this because WH requires for stage 2.

EAO: interested in how other libraries implement. Having a hard time understanding how this is done in ICU.

SFC: Anba has posted C++: https://github.com/tc39/ecma402/issues/398#issuecomment-4136166749. Not sure about other languages. CLDR/ICU has taken this problem head-on. ICU modeled mixed units as single unit string w/ scalar. Inclination is – pros and cons exist – advantage of using scalar approach outweighs negatives.

JMN: How does this impact interchange? Scalar feels better, bag feels off.

EAO: Would foot and inch be used in interchange?

SFC: Question: mixed unit Amount, how to represent?

EAO: Amount instance would have .value that's an object {foot: 3, inch: 7}

SFC: And then toString?

EAO: Haven't thought about it.

SFC: Okay, you have one of those amounts, you call .convertTo?

EAO: You don't, because CLDR data doesn't support foot and inch as source. Target, not source.

SFC: so should throw because CLDR doesn't support it?

EAO: current spec language it would throw. Not that much text to force conversion. Highest to lowest for example.

SFC: expectation: having such a value and calling convert, take all units in unit identifier, figure out target unit, convert each, add together.

EAO: Could work.

SFC: Now look at Amount constructor, created with unit foot and scalar value of foot, say 5.5. That creates Amount w 5.5, unit foot. Now create Amount w/ foot-and-inch, give it a scalar 5.5 as input. What happens?

EAO: Failure. - and - means we're looking for a value object, what you've given is not value object.

SFC: Value object, field foot but not inch, what happens?

EAO: Fail, we're converting from `undefined`, not a supported value.

SFC: You have that, your object has foot and inch fields, but also other fields like meter. Drop the meter?

EAO: I think so

SFC: Finally Intl.NumberFormat configured with foot and inch, object passed to .format is object with fields foot and inch, do we support this operation and read fields foot and inch from object passed to NumberFormat

EAO: yes, but first check that it follows unit protocol. If fail, use unit fields. If unit protocol, check same thing from value field.

JMN: object seems like we're exploring new territory

EAO: have this conversation in plenary? Amount depends on this, we can prepare ground – if it's approved, we can propose Amount based on this.

SFC: If structured object as value for mixed unit, expectation that this would align with making the unit being an array of [foot: string, inches: string]. If structured on value, structured on unit too. If flat on value, flat for unit

EAO: Prefer not to depart from CLDR unit syntax.

SFC: less a departure, more a typesafe version of that. CLDR specifies mixed units as being grammar of a bunch of single units concatenated with -and-. Follow entire CLDR grammar up to the concatenation.

EAO: we already have -per- support in language, if we go with non-flat string representation for -and- equivalent, I'd be surprised if -and- were replaced with object, while -per- would still be always a string.

SFC: mental model for why structured: makes sense that strings in arrays are keys for map in value, otherwise must parse unit to find keys, seems unnecessary/problematic

EAO: already doing that for -per-

SFC: We don't need to do that?

EAO: in intl.numberformat. Or do we delegate to ICU and ICU does it. Allowed -per- when checking for valid unit identifier, other than that we don't do anything.

SFC: that line in spec is to elide listing all of the units out, like a comprehensive closed list, we don't, we say here's the 45 single units and we can combine them with -per-. A spec shortcut, not parsing.

RGN: same true regardless. We have finite set of sanctioned units and combinations of them are also finite.could be infinite if multiple -per- allowed, but we don't.

SFC: plan: write a more fleshed-out writeup of what structured value field looks like, encourage comments. Even if scalar, useful to document why scalar. Only way to do that is to explore design space of structured.

EAO: probably we should strip out 402, since now part of intl unit protocol. Problematic to have same spec changes in two proposals.

JMN: sounds fine

SFC: I don't think intl.numberformat needs change here in Amount proposal.

EAO: eventually changes to 402 due to moving content from 402 to 262

Next steps

- SFC write out explainer field

2026-02-26

Attendees

Eemeli Aro (EAO)
Ben Allen (BAN)
Jesse Alama (JMN)
Shane Carr (SFC)

Agenda

- Amount: Support for measurement systems and measurement unit overrides? (context)

Notes

Amount:

JMN: Are we going to ask to Stage 2?

[significant discussion resolving in “yes”, we solidly meet the Stage 2 requirements, some TODOs / open questions remain (what about all the spec text copied from 402?) but not enough to prevent the ask for Stage 2]

EAO: Amount.value: two reasonable options: toPrecision or toExponential notations that already exist. Okay with both, preference with toPrecison

RGN: Advocate for scientific notation. Unambiguous, unlike integer with 1 or more zeros

JMN: +1. Part of current iteration, or leave it for future?

EAO: leave a TODO

RGN: pref for test-first approach, could justify TODOs

SFC: strong preference with scientific notation everywhere

JMN: resonates with original name
Intl Unit Protocol:

EAO: Move all the Intl parts of Amount into Intl Unit Protocol? Advantage: makes Amount largely 262, put all Intl stuff into Intl Unit Protocol.

Next steps

Amount:
EAO -> prepare slide skeleton by Friday
BAN -> iterate on slides on Friday, prepare to present at plenary
JMN -> update spec to [insert here]
? -> make PR adding ask for Stage 2 to plenary schedule

2026-02-18

Attendees

- Jesse Alama (JMN)
- Olivier F (OFR)
- Eemeli Aro (EAO)
- Richard Gibson (RGN)
- Shane Carr (SFC), from 30 minutes

Agenda

Notes

OFR: Let's align with JS. I don't see "float" in this proposal (speaking of Amount).

EAO: Mathematical operations are all done with JS Number. This introduces some rounding. Converting between feet and inches is easy, but feet-to-meters introduces issues. I'm trying to minimize the amount of data to be carried in a browser, while sticking to the language. We multiply or perhaps divide. That leads to: what do we do with rounding? Yet again, take something from the intl spec, which does have support for rounding. convertTo would then allow us to move some 402 content.

OFR: rounding after conversion?

EAO: yeah

OFR: what I'm worried about is a number not representable as a JS number, but we have to forceit into a JS number.

EAO: we don't want to solve that problem. We weren't able to identify a use case on amount conversion where the values weren't representable as JS numbers. We can imagine cases involving interchange where precision needs to be more than what Nubmer provers. But for conversion..

OFR:

EAO: we do not support values beyond precision/range of number. Nothing more than what we can get out of serialized numbers. Using that as the public value representation. How it's done is open to discussion.

Jesse: To rephrase this point, it's a reduction of the scope, because we're going to eliminate the idea that we're going beyond Number, somehow. There were earlier iterations of this discussion where we were hinting at that, but that's not where we are today. Numbers, or at least the string serializations of Numbers, is the home base for Amount. In earlier iterations we said things like arbitrary strings, or Decimal, things like that, way out there, but now we're grounding it in Number.

EAO: For unit conversions? We're still providing for the representation of numerical values that are not representable with Number

OFR: How does it go together? What happens if you have a conversion on an Amount that's not representable as a number.

EAO: You can do rounding, you can do Infinity, you can do zero…

OFR: But you do have a round trip, you can parse that other Number back to a Number.

EAO: Possibly yes, but the spec language needs to allow for an implementation not doing that roundtripping. One thing I said in PR 81 is I'm presenting value as a direct property of the thing, but it should almost certainly be a getter. In the spec we could say it's a mathematical value or a string representation, but an implementation can just keep the Number it was given originally, so it doesn't need to roundtrip that way. The space here for us to make sure the spec allows for this, but doesn't need a [missed]

OFR: The value slot is tied to number, tied to BigInt, tied to string, and we do three different things when we pass it?

EAO: No, that's the other proposal that SFC has submitted, how Intl.NumberFormat does input, as in SimplifyInstanceAPI, yes, the part you're referring to is referring to another proposal, the Intl unit protocol, and this is describing how we change Intl.NumberFormat so when you call .format on that, instead of looking for an Amount instance, it says "it's an object, does it have a value field, does it have a unit field maybe, then it behaves how we want it to change." But according to PR 81, the value(?) would be a string.

OFR: So the default is returning a string, if you pass it to numberformat and do rounding we would need to go back to a string?

EAO: If you pass a string to Intl.NumberFormat.format, it converts it to an Intl mathematical value. Even now, in Intl.NumberFormat we support numeric string inputs that have higher precision than is supported by Number. To some extent we already have this string-like representation of Numbers in 402.

JLM: I like the hoisting parts into 262, I think there's lack of respect for the work that's been done, but is hiding in the 402 silo. Rounding stuff is there, it's impressive, 262 doesn't touch that. Question: do we want to support all the rounding modes

EAO: I don't care about rounding mode, I've never touched it for stuff like this. Anything is acceptable to me, so long as it doesn't expand beyond 402.

JLM: Does CLDR recommend any particular type of rounding?

EAO: Do we have text in 402 what the rounding modes mean. We rely on ICU effectively, since that text corresponds with what's in ICU.

JLM: Well, more like having a standard default and then optionally allowing these other ones.

RGN: We end up there naturally, 754 is pretty clear about halfEven being the default, 262 tends to use that unconditionally except for a few cases where it truncates unconditionally.

EAO: I would be fine going with halfEven always.

RGN: But NumberFormat allows you to specify it, when you're configuring a Number formatter it should be there

EAO: You have an input, it's going to start from a string representation of the number, it doesn't need to be provided at an earlier change.

RGN: I think if you're specifying a limit on precision via sig digits, number formatting comes along for the ride, but only where it's specifiable. 402 lets you specify the mode, probably this does too.

OFR: My main concern is we're adding a new number in general, the rounding is not the issue. If you can give an arbitrary string that represents a number, I don't understand from a design POV why you would want a numerical object that contains a value that's not representable as a number.

EAO: Well, BigInts can already represents values that are not Numbers. not having a string representable of a number that's not representable as a Number or a BigInt

OFR: It's justifiable in intl, it's strange to have otherwise.

EAO: Why is this a problem

OFR: You're building something that you can't represent as its pieces. Parts you can represent, parts you can't

RGN: The language has a concept of numeric string, it just doesn't get its own type. In conversion functions. We have StringNumericLiteral. To be precise: there's the one that applies at the level of parsing source text, different one where there's literally a language string and you have that when converting to a number. Including unary + or passing to the Number constructor. the concept is implicit in the language, though not explicit, the type is still string but there's strings that are used as a number. You can do array functions, .at or .slice, string goes through operation to convert it to a number. In such cases the user code never sees an instance of Number, but internally the input string passes through this machinery, in spec terms, and is then treated numerically once you do so. That's really common, most of the APIs that want numbers will accept anything, and when giving a string or something that produces a string, it passes through this.

EAO: Intl.NumberFormat does the same too, but allows through Intl Mathematical Value values that aren't representable as Numbers

OFR: At that stage it makes sense to me, I just find it interesting/strange to introduce this new number type as something we give to the user.

RGN: it's promoting a concept that already exists. You have numeric strings, they're just not a type. The value associated with an amount can be typeof(Number), typeof(BigInt), or this subset of strings, arbitrary decimal strings, which is a narrowing of that subset. This already exists elsewhere, it's just Amount is rejecting strings that don't conform with that grammar. We need to make sure it conforms exactly with Intl strings taken with Intl.NumberFormat

EAO: It's open where to construct an Amount – hex string should work, but .value would be in decimal rather than hex, always.

RGN: And iof you give it an overlarge hex string, is that lossy?

EAO: It'd do whatever we're currently doing. No new behavior. I want to distinguish strings that contain a numeric value from strings that do not, so that we can make use of them more effectively. Is this answering some of your concerns, OFR?

OFR: It doesn't really. I see the point, I see where you're coming from, but then my reply is "but then it's an Intl.Amount, it's not an Amount in JS. Conversion is floats, why isn't Amount a float?

EAO: We have use cases not about formatting – interchange. We don't have a capability to represent all JSON content without lossy processing. JSON capable of supporting numerical value with infinite precision, you JSON.parse and you get a Number. Use cases for numbers with higher precision than Number, most of Decimal is about this, that amount can be used for those values. I would be fine with Intl.Amount, we have to recognize we have utility beyond formatting and representation, it will be used by developers, are we comfortable adding something that can be used for a non-intl purpose

OFR: You're effectively putting values into a string, you're stuck there, you can pass it to NumberFormat, you could do that with a string, other than that you can't do anything, you can call .convert, that's delaying the parsing/rounding, what else do you gain?

EAO: You can use an external library like decimal.js that add capabilities we don't have

OFR: Can't I just use decimal.js?

EAO: You face the issue that you're stuck with that library. No interchange way of using values of higher precision than Number if we use dfferent libraries.

OFR: Effectively interchange format with Amount is string, not super useful for Numbers because it's slow.

EAO: it's only a string if you look at it. The intent is an impelmentation can internally store it with something more performant. I'm fine with Intl.Amount, are we okay adding a thing to Intl that we can recognize have non-Intl uses. Scope-creeping all of 402

OFR: Scope-creep to expose a new number format. If you need to do something that's not number formatting, it will effectively be slower, implementation would have to recover that performance.

OFR: Intl.Amount not a great solution. Prefer introduction of new things that can live in the value slot elsewhere, don't mix it in this discussion.

EAO: So you think value should contain a Number or a BigInt, and that we ought to consider what's actually in there later in the design stages? If we go for example with the current API proposed in PR 81, typescripty discussion of what's there, leave what's actually in value for later? Have that discussion at stage 2?

OFR: That would make the discussion easier.

EAO: I would be happy with this plan.

JML: OFR makes a good point, the use cases are kind of self-referential, they rest on intl.

BAN: what about implementations without Intl?

EAO: Some of the spec will be in 402.

RGN: anything that relies on locale is 402. No way to even parse that in 262.

OFR: Good argument for it being decomposable, intl spec can – there needs to be this interface where Intl can query it, but it needs to be described in 262 terms, assume very little about each other as possible

EAO: that's part of SFC's intl unit protocol

SFC: Seems important for a stage 2 proposal, it's important to have an idea of what is the type of number value that goes into the field of amount. I had been pushing back on making it a polymorphic number, we came up with string because it's the superset of Number/Decimal/BigInt, if thing we can get advanced is polymorphic, Amount type agnostic to what's inside it, then maybe that's okay. If you do call a .convert function on that, does the inner type get converted to a float? Or do we attempt to implement conversion in the same numeric space? Do we want an amount that doesn't care about which numeric type?'

EAO: OFR says: first have the discussion "should we have an Amount", then at stage 2 we then figure out what the value is being presented out of it. SFC, my understanding of what you're saying is we need to resolve this before stage 2. I'm more on OFR's side: separating these discussions would make it easier.

SFC: Not allowed to advance to stage 2 without a spec, can't have a spec if we don't know the type

EAO: [differs, quotes stage 2 requirements]. Uses cases we're imagining are interchange and formatting, formatting is going to get the string value, interchange is eventually going to get the string value if it needs to go somewhere else. Other operations can work on something other than a string representation of a number, we don't have agreement on how it should work. If there are other use cases that we want to consider at this higher level, we should identify/enumerate those

SFC: We can try this – I think we're going to get pushback about trying to advance a proposal that's not very specific. Number type is a major detail. If that gets to stage 2, it's easy sailing from there.

OFR: It's good to know what's in that slot. My main point is that there's two separate discussions, and we shouldn't mix them.

EAO: would you be fine with either a bigint or a number or a string?

OFR: I think – it might lead to a better outcome. Where you can guarantee that if you stick a float into it you get conversion on exactly that float, without needing roundtripping.

EAO: We can already make sure that's the case. Effectively if you go from a float to a string and back to float again, you'll get the same float. For values within the Number range. If we provide accessor for non-string values, we need accessor for fractionalDigits/significantDigits as well, because it's an aspect that Number doesn't hold.

JML: +1. Do we allow the polymorphic type.

OFR: at least it makes it clear is that if you put a string there the only place you'll get anything interesting to happen is NumberFormat, otherwise it would break.

EAO: and if you're using decimal.js and other userland libraries, you can do all sorts of stuff. Q: does decimal.js contain a .value?

SFC: If it's actually a polymorphic amount, if that's what we can advance, then we should allow userland numeric types there. The question is the semantics of .convert with these in. These are concrete questions that could have concrete solutions.

JML: decimal.js returns a string

BAN: Are there stakeholders who might object?

EAO: We've only discussed this in small groups.

EAO: If we get to advance it like this (type agnostic) see if they can present a use case for supporting something like an object of whatever type there, or whether –

SFC: If stage 2 is easier to obtain by just supporting builtin types, then that's enough major semantics, that's fine, and do we support userland numeric types, that's a narrow enough solution to have later.

[...]

EAO: As in PR 81, prefer to drop precision fields. Can be deduced from numeric string value.

SFC: Main functions: setPrecision, .convert, .format, let's look at .setPrecision – let's say we have prototype.precision…

EAO: in the constructor, yes

SFC: regardless, it can be implemented on the Number type by calling Number.round, maybe we don't support rounding mode, store the precision separately, can be implemented on that, just using numeric types, use Number value, get Number behavior, use string value, precision stored in string but presented in precision field. BigInt doesn't have trailing zeroes, precision always fractionDigits: 0.

EAO: What's the problem we're solving? We can get everything to work if we say if you use precision, you have a string.

SFC: .convert should have Number behavior.

EAO: If we're talking about digits, we need to effectively format number into a string – maybe not literally a string, but get the values you'd get in the string, you need to work with the digits.

SFC: .setPrecision

EAO: The value you end up with is always a string. Even a BigInt, even a Decimal.

SFC: Seems a little goofy, depending on how you construct it and what functions you call, it can be converted to a string in a way that only the spec authors understand. But if it gets it advanced, it gets it advanced. But I'd expect a Number amount to remain a Number amount… it helps if it's immutable though… Amount is basically a protocol with a field of a certain shape, with a prototype. We don't have anything like that in the language. Just a protocol for an object shape with frozen fields, with a prototype with callable functions

EAO: Feels like something that exists somewhere. Temporal values are pretty much this.

SFC: Temporal values are more opaque, but yeah, a little bit.

Next steps

2026-02-12

Attendees

Agenda

- Decimal via BigInt https://github.com/tc39/proposal-decimal/issues/202
- Intl.Amount

Notes

Amount

SFC: Rename to Intl.Amount?

EAO: As a response to V8 concerns, would be better if we could point to something as evidence. We don't have a public discussion of this. This appears to be a solution looking for a (documented) problem.

SFC: Renaming responds to a couple of issues in the repo. #71, #72, #75

EAO: So, no interchange?

SFC: We've had this discussion before. If we call it Intl.Amount, it's close to Intl, not intended for arithmetic.

EAO: So we retain "intended for interchange but it's intl". So this helps?

SFC: Yes, though though they need to see the whole proposal first. If this gets to stage 2, we can propose renames. We've proposed a TG5 study but no traction yet. With "Amount", people will knock on the door forever asking for arithmetic. "Intl.Amount" addresses that concern. Other names might be better. "AnnotatedNumericString", e.g., but I don't want to put that in front of plenary. Let's get semantics to stage 2. Another advantage of Intl.Amount: WH's concern about unit conversion. "Same as what we propose for Intl."

EAO: What strategy? Or are we saying "whatever Intl does".

SFC: Unit conversion will use the same strategy.

EAO: We couldn't think of a use case where precision of amount after unit conversion isn't greater than that of Number.

https://github.com/tc39/proposal-amount/issues/43#issuecomment-3759228091: "If Amount becomes part of standard JavaScript, math.js would likely move toward having its Unit facility based on Amount, with the dimension attributes and arithmetic operations overlaid, although Amount being tied to one specific numeric type (or having just two flavors Number.Amount and Decimal.Amount each tied to one type) would be of some concern, as we would not want to drop support for rational and/or complex-valued unit-denominated entities."

EAO: HAppy to go with a somewhat opaque (Intl).Amount. Introducing Intl.Amount even with acknowledgment that we expect that this is to be used to interchange isn't bad. That makes it

easier to keep its definition in one place. There would be interest in unit conversion that respects locale. Need to work on the spec text, especially the unit conversion stuff. That's must-have. In particular, an explicit list of unit multipliers, or do we just refer to CLDR? If we call it Intl.Amount, the alignment make sense. But we need to have this written up. Who among us can do this? I can sketch an API design, because it's close to what I sketched a while ago. We also sketched out the unit conversion.

SFC: One issue with unit conversion: mathematical values or Numbers? One advantage of "Number.Amuount" would be that we're working with Numbers.

EAO: That's a solution to an imagined problem. We need to see that solved. I think we got alignment on the precision already.

SFC: Not about precision of output, but how arithmetic is done. We might be requiring engines to ship a decimal library if we pick mathematical values.

EAO: But I refer to my comment from last time. Unit conversion should be fine if we need Number.

SFC: 2.1 or 2.2 depending on how we understand the semantics. Number.Amount has been growing on me.

JMN: What about arithmetic in Intl?

SFC, EAO: just powers of 10. Engines can do that.

EAO: API spec text: constructor…do we provide .toValue to get the number?

SFC: Just rely on toString, parse out any unit.

EAO: How does this align with intl unit protocol? Does it expect a value and a unit or other fields like that? Amount should match that protocol?

SFC: Presentation was a general temperature check. If we advance Amount, intl-unit-protocol should come after. I'm fine to wait till we get to stage 2.

EAO: My preference is .value would be numeric string. Then there's no .toString.

SFC: Or do you mean .value ot ToValue? Does that include annotation, precision?

EAO: what is that?

SFC: Rational, margin or error, unit, … IF we have just toString that gives all annotations,

EAO: Where did we end up in the discussion about losing precision? If we do .with losing precision, then gaining precision?

SFC: round on the way in

EAO: the .with could impose trailing zeros. I'll work it our and make a PR for you all.

EAO: We need two conversion methods (one locale sensitive, the other not)?

SFC: LocalConvert takes a usage and a locale.

Keep trailing zeros

EAO: RGN, your comment about test262. I have to look in my format.js polyfill or does the spec text produce the wrong result.

RGN: Not clear to me either; just looking at test assertions.

EAO: Right. I think the test assertions are wrong, but I'm not sure why yet.

Next steps

- Add unit conversion back to amount? Rational annotations, too
- Propose an API structure for (Intl.)Amount.

2026-01-29

Attendees

- Eemeli Aro
- Richard Gibson
- Shane Carr

Agenda

Notes

- Eemeli: Is there a path for Decimal?
- Shane: Should rationals be supported in Amount?
    - Eemeli: It sounds like they are in the same category as the precision field
- Eemeli: I think all the formatting use cases of Amount are covered by the space represented by Number
- General agreement that the proposal should be updated to include unit conversion and rational annotations, with both highlighted as potential discussion points in plenary

Next steps

2026-01-15

Attendees

- Jesse Alama (JMN)
- Shane Carr (SFC)
- Ben Allen (BAN)

Agenda

- Be more clear on champions for each work stream

2026-01-08

Attendees:

Agenda

- Discuss pros and cons of these two paths: Number Amount vs String Amount
    - https://github.com/tc39/proposal-amount/issues/75
    - Jesse: Would Decimal.Amount be motivated if we had Number.Amount?
    - Eemeli: I'd expect math operations to work on Number.Amount, which is a new direction.
- Plans for January plenary
    - Eemeli: I'll be out for the next 3 weeks

    - Eemeli: Seems like people are onboard with unit conversions and that's a direction we should pursue

Notes

JMN: Number amount vs. decimal amount: use cases?
EAO: Use cases align with use caes for decimal itself
SFC: commitment to number.amount would signal idea of having amounts for numeric types in general

Next steps

- Two trailing-zeros PRs in SFC's review queue
- Issues about intl-unit-protocol to be addressed in TG2
- Intl-unit-protocl repo is still in Shane's GH profile

2025-12-11

Attendees
Agenda

- Riffing on Amount next steps
- SFC: V8 feedback: "if amount is added with Intl.MV inside plus unit conversion, then we have basically a new data type doing math on strings." A solution proposed to Oli: an amount backed by different numeric types: NumberAmount, DecimalAmount as separate types. Conversion works with JS Numbers and outputs a NumberAmount. Easy to specify, reason about, and test. A StringAmount/IntlAmount wouldn't support unit conversion, but would support rounding. Supports scientific notation, roundtripping to a string, formatting. NumberAmount <-> IntlAmount, but you wouldn't need to. Library can decide what to accept. IntlAmount if they're doing mainly formatting. This might make it Oli happy.

Notes

EAO: seems like this proposal puts implementor concerns above user concerns.

SFC: The concerns that are real, and could be addressed e.g. if precision is required. Oli might say that this is motivated by performance (string → float conversion). By having StringAmount, we could have unit conversion, in the spec it would just go to NumberAmount, and then back. If we hae a StringAmount we could also say that we do calculation in decimal space. But they might say "why not have decimal as a basic type?". I might agree that this is implementor over user concern, but not an invalid question to ask. And for decimal, even Decimal128 wouldnt be enough, we'd need BigDecimal. By having a NumberAmount and conversion to StringAmount by requiring a precision, rounding errors ought to go away.

EAO: If we require precision, … which we can set max on … we can do this safely.

SFC: I personally am ok with that solution that EAO just described. But Oli's concern is valid.

EAO: Could we get this in a GitHub issue?

SFC: I did sort of do that.

Next steps

- Review issue 12 in keep-trailing-zeros
- Review issues in intl-unit-protocol https://github.com/sffc/proposal-intl-unit-protocol/issues

2025-11-13

Attendees

Agenda

Notes

Next steps

2025-10-30

Attendees

Agenda

- Keep-trailing-zeros: new PR https://github.com/tc39/proposal-intl-keep-trailing-zeros/pull/10
- Very small values: the rounding of very small values

Notes

Next steps

- Plenary deadline is Friday, November 7th!
- Potentially clean up spec text in 402 around “logarithm base 10 of x” (e.g. ComputeExponent)

2025-10-16

Attendees

Agenda

- Keep_trailing_zeros:
    - w3c i18n review complete
- Amount:
    - Tpac breakout session for html amount proposal, EAO will have call w/ Mozilla people on Tuesday, surety that what we’re doing with tc39 Amount is in-line for HTML amount
    - Global object motivation: “who is the proposal for?” -> more than just Intl?
        - V8 team sees need to prove interop use case. They feel that effort on BigInt was wasted. Worry about performance if users convert numbers in/out of it. Will people assume they can do arithmetic on this type?
        - EAO: drop interop as use case, make it into Intl.Amount. In that situation we’re in the same space as table formatting, acknowledge that there are parts of Intl useful outside of Intl, bake in affordances to make sure that stuff doesn’t just fail – so that ways are provided to abuse Intl stuff in the way that JS users abuse anything they can get their hands on
        - EAO: the name Intl.Amount is better than other names at convincing developers not to do arithmetic with it.
        - RGN: I think it’s a bad choice, and will preclude extension. (i.e. never support arithmetic)
        - SFC: The v8 team explicitly wants to preclude arithmetic.
        - SFC: argument for protocol-only alternative worse. Let’s say we have an annotated string. When we get to the point where it’s used for interop, we could introduce a TypeScript type called NumericString, we go to React, we go to libraries that will interact with these sort of things, then people have NumericStrings floating around, you can format them in Intl.NumberFormat, but now you’re in a situation where Strings are used in the data type and there’s nothing you can do to stop people from using + and < and so forth with them. If we use Amount, we can make those

operators do the right thing or throw. If we just use a String as the *lingua franca* for this type, we can't do that.

- EAO: Strings aren't useful as an interchange type, but we need to give arguments against an Object interchange type. Strongest argument I've heard is no guarantee of immutability, but that feels like a weak argument to make here.
- JMN: Apropos immutability, KG expressed concern about making Amounts immutable, was fine with them just being frozen. SFC's issue 71 bumps up against this question of immutability.
- SFC: Suppose we have a world where we have the objects that Intl formatters format, and then encourage in Intl the use of those objects as more than just short-lived things for formatting, i.e. used for data storage and interchange. Five years from now this becomes the standard for dealing with numbers paired with units. Can we communicate clearly why this outcome is worse than an outcome where we have the intrinsic?
- BAN: If it's just an object there are no guarantees that the details about e.g. significant digits are correct.
- Significant discussion of establishing appropriate/clear communication channel w/ v8 committee
- EAO floats idea that this should be handled on github
- JMN proposes participation in v8 community call
- RGN: "Strip it down to parts, look for motivations of each of them." It's a struct with specific fields that can be used to convert between numerical strings and numeric types (?)
- SFC: The protocol is a record with two fields. "number" and "unit", type of "number" is any numeric type accepted by Intl.NumberFormat, "unit" is a String. If you want precision/significant digits, the "number" field should be a String. Do we want a function that gives a number and a precision and it gives you a String? Because if so, that motivates Amount.
- EAO: strong preference: field "value" that always has a numeric string representation of the value.
- RGN: Precision?
- SFC: Precision is stored in the string, so long as you started with a string.
- EAO: Number.toPrecision gives you a string with the precision you want
- RGN: And it's essentially on the developer to take their numerical value and put it in the correct precision.
- RGN: In the minimal form, a number would be converted to a numeric string on input.
- SFC: If there's a bare object, where would the constructor be?
- RGN: We're still talking about the intrinsic. In that case, we must return strings as the numeric type since they encode the precision, which would otherwise be lost.
- BAN: How about NaN

        - RGN: Yeah, it would include NaN, Inf, -0, but that's a different discussion. The shape doesn't change.
    - New issue: https://github.com/tc39/proposal-amount/issues/71

Notes

Next steps

- Deadline for submitting materials for plenary: Friday, November 7th
- Decimal: refine README, won't present in November plenary (waiting for Amount to get to stage 2 first)
- Still outstanding: message to KG about concerns re: primordial vs. protocol
- Still outstanding: message to MM about accessing internal slots (should be easy)
- Next week: V8 contributors call, will ask about Decimal
- Task for Eemeli: constructor without being an an intrinsic
- Talk with Jordan about intrinsic motivation. Invite to the October 30 call?

2025-10-02

Attendees

Agenda

- Strategy: what we learned, what are the problems, what could some solutions be, unit conversion, what are next steps
    - Protocol: Why not just a protocol
        - Eemeli: if we get Kevin onboard, is that a strong signal that we have alignment from other stakeholders?
        - Shane: I think so
        - Action: Ben to follow up with Kevin
    - Protocol Slots reading
        - Shane: Where are we currently on this and is there anything else to do on this?
        - Eemeli: I think we are currently agreed that there shouldn't be a need for slot reading
        - Shane: That's fine for now; action in my mind is to get a writeup for what exactly the constraints are for when we can read from slots vs when we must read from protocol over membranes (red/blue). Ben to follow up with Mark Miller.
    - Protocol: Object vs String

        - Eemeli: If we go with the string protocol, things become easy. If we go with a protocol as an object with functions/properties, it creates a need for us to ensure the Amount object satisfies that protocol.
        - Shane: Object protocol seems more modern than string protocol. Could be just plain prototype functions, or could be like [Symbol.toAmountString] that goes back to a string.
        - Eemeli: Probably we'd want two functions, one for the numeric value, one for the unit. The numeric value should be a string. Maybe just properties "value" and "unit"?
    - Naming
        - Discuss comment from V8 team about over-generality
        - Eemeli: Is interchange a use case? If it's not, then Intl.Amount. If it is, then whatever name we change Amount to ought to cover that use case. If the use cases are formatting and interchange, those are sufficiently different that a name at the intersection is going to be generic.
        - Shane: WDYT about AnnotatedNumericString?
        - Eemeli: It sounds long enough that the implication is that you shoudn't be using this.
        - Shane: This is to make it clear that we're not adding a new numeric type. Maybe just NumericString is sufficient?
        - Eemeli: My preference is Amount with limited functionality
        - Shane: I know, I'm just trying to bridge the gap.
        - Shane: Maybe a user study would help. I don't want to decide this based on vibes. I want to quantify the vibes.
    - Numeric conversion functions
        - Eemeli: It seems like they're blocking now because they don't want to block a future proposal adding arithmetic that doesn't exist.
    - Remove significantDigits and fractionDigits?
        - Shane: I haven't heard anyone pushing back on those properties
        - Eemeli: What is the use case? It's not clear that either of the use cases need these properties.
        - Shane: It could be useful if you're writing a custom formatter
        - Eemeli: But you can get that information from the string
        - Shane: I'm fairly neutral. We should do whatever helps us get the proposal passed one way or another.
- Intl Keep Trailing Zeros
    - Eemeli: It's on the agenda at the W3C i18n WG.
    - Eemeli: The W3C i18n WG seems interested in discussing new feature development like <amount> and message bundles.
- Amount (JS + HTML?) session at TPAC?
    - Eemeli: I need to sync with Tantek.
- MessageFormat use case
    - Eemeli: "oh, I mean the the the story there is effectively that the with something like an amount you can do the DOT format call and pass the information there.

And because message format. With message format, there is no constructor options bag where you could pass that information as a secondary place. So it would Be quite clumsy. And and furthermore, we have not in draft because colon unit and and colon The colon currency. are draft and not stable in the, the Unicode spec, but those do specify that and implementation should Support. Inputs where the, when you formatting our currency value or a unit value. That those that comes to the format in a single something. So that at least a protocol like this exists."

- Eemeli: "And and the the user code part of that does come into play as well. Because message format allows for the and and invites, the definition of custom custom functions to use in messages, which get as inputs. The values that are given on the outside of the message. And this sets up a case, where a A function might be formatting a value, that comes in whatever goes into a dot format call for internal number format effectively for when it's dealing with a numeric value. So it needs to have a way to to be able to deal with. And an amount effectively. As well. Oh that, that is an interesting. Sorry, the D You use case that that presents from there to amount. Is that if we have, Some input value that is a number or a bigint or a digit string or an Amount instance. it would be useful to be able to, Construct, a new amount around that. And and to be to, then be able to have the value, always be a digit string. So that you could use the the amount. Constructor, as a way of normalizing, the representation of numeric values into one."
- Eemeli: "That. Who was it? That had concerns about constructing. A new amount from an amount on one of the issues where there was a discussion on this. I can't remember, I was On on the, The Proposal amount repo. That did not really. I think I discussed in committee. actually, yeah, that sort of Value, numeric value, normalization. Is kind of the use case. It's kind of done. Alright, already by just stringifying whatever. but, That doesn't ensure. That the what you get is actually a. Digit string."

Notes

Next steps

2025-09-22

Attendees

Agenda

- Shane shares' feedback from V8
    - V8 doesn't want to proceed in a direction that evolves into a third numeric type
    - Fear that we'll put in arithmetic in the future
    - Ongoing use case concern
    - "formattableAmount" name
    - "Use it as input-only and then use it for your output in intl formatting, that's acceptable"
- Shane: How are we going to make it future-proof?
    - Eemeli: We could try to support error bars in the future
    - Nicolo: We could have another property at least for rounding increment
    - Shane: I like the extra property. For the string pattern, we could have "annotations" like for Temporal string format.
    - Shane: Maybe same mechanism for rationals. Or I'd think you'd have a / in the string. But maybe it's better for the numerical value to be a decimal representation, and have the exact rational as an annotation.
    - Jesse: about the thermometer. It makes me think about unit conversion. X degrees +- 0.5. But what about mixed units, like 1kg +- 2g.
    - Eemeli: I do not think we should build Amount to support everything now. Just to potentially support other approaches to precision and numbers in the future. So the string representation is a question for later.
    - Eemeli: No need for it to do anything novel – wrapper for numerical value
- Shane: Naming:
    - Shane: making clear that it's intended for the version of Number that's intended for stringification / not usable for arithmetic. Believe name "Amount" signals it does something it doesn't do / will encourage developers to use it for arithmetic. Something like "FormattableAmount" makes it clear that it's just for formatting and not for arithmetic. We have "prior arts" slides, but those include languages that do math. We're not introducing a numeric type
    - Eemeli: Agree that name should reflect intended use. Concerned that specifying it to be FormattableAmount or placing it under the Intl object is leaving out interchange as use case. If we're leaving out interchange, we should be explicit about doing so. If we're leaving it in, FormattableAmount or Intl.Amount are communicating exactly the wrong thing. Is interchange in? If so, Formattable and Intl. are wrong for this
    - Nicolo: something like AmountResult?

- Eemeli: also blocking out interchange
- Jesse: Other names considered: Dimension. FormattableAmount seems to take the wind out. LabeledNumber? UnitNumber?
- Nicolo: for v8 team, would it be enough to say in slides that champion's position is that we're looking for a name of the proposal that's indicating that it's used for formatting? Say we don't have one picked yet, give the options we've come up with.
- Shane: Calling it Amount (placeholder) seems to be okay. V8 team mostly on-board for the Intl use case being enough, we'll go along with adding the type, make sure you do all these other things correctly. Calling it AmountPlaceholder is for Stage 2 purposes something that we could do and largely still be cognizant of their concerns
- Eemeli: But are we still including interchange as a use case for Amount? That is the core driver that needs to be determined before we start picking the name.
- Shane: Yes, I want interchange to be part of the value proposition. Intl.Amount is a proposal that I would support, but interchange makes the proposal much more strong. I'm on board for that / I think we as champions should continue advocating for that.
- Nic: Intl as subset of interchange. I struggle to think of how if we drop the interchange use case the API would change. Say we only do formatting, how does the proposal change? Only way to think about is that this only has a format method on it.
- Eemeli: If interchange is in, then is what I think we've arrived on. Then we mean Amount is a thing that we could use to grab a value and then use it. Can't think of name that includes interchange and keeps arithmetic out. We're in a space that is in the space of arithmetic, if interchange is in the name ends up being something that will to some readers imply that arithmetic might be in scope. Here I see that's a valid use case in Amount, it's just not valid for JavaScript spec to define that how that arithmetic works. If we drop interchange, if it is a formattable thing, then it should be in the Intl namespace, therefore Intl.Amount, we drop toNumber, toBigInt, toDecimal, we keep whatever's required, like, a bare toString for the numerical value.
- Shane: Think V8 / other delegates would approve tomorrow Intl.Amount, it's an Intl thing, I'm convinced by the intl use case, it's narrow, it's scoped, it's fine. I consider that a plan D, though, there's still room for us to strike the right balance, this is pushing the V8 team a little bit because they are skeptical, it's a little bit of an uphill but an uphill that's surmountable. Smaller hill than full Decimal.
    - Name ideas:
        - Numerics.Amount. Puts it into a scope, bringing things into namespaces seems to be nice. Namespace Numerics lets us add additional things later, though we're not saying it now. Shouldn't be scared of single-element namespace.
        - PlainAmount. "Plain" doesn't mean anything, inspires users to look up what it means. Sort of like PlainDateTime

            - AnnotatedAmount or AnnotatedNumeric or Numeric.Annotated. This emphasizes what it is.

    - Eemeli: Numerics.Amount even more than everything else implies that it can do arithmetic. This would be easier if we could have it with the V8 people rather than mediated through Shane. Need to get these concerns filed as issues at some point.
    - Nicolo: +1 to Numerics.Amount sounding like math. FixedAmount?
    - Nicolo: We should summarize this discussion in the continuation. Shane discuss V8 points.
- Additional feedback from V8: how does this interact with <amount> html tag.
    - Eemeli: What's the desired shape of the answer? <amount> html element is barely an explainer, not much of a there there for us to match.
    - Shane: That's more info than I had about it. *If* there's an <amount> tag in html, then if you read it do you have an Amount? Is w3c going to be okay with us making this decision. If <amount> doesn't return an Amount, what should it return? Should we choose a different name?
    - Eemeli: Author of that explainer is in favor of us calling this thing in JavaScript amount, then that's not a concern.
    - Nic: contents of <amount> element is just the formatted amount, the reason to put it into <amount> is that when you build the semantics of the page you can see that this thing is actually 200 dollars, even if it's not formatted in that way. Supports currency as unit? Doesn't consider sig digits, it's fine if our proposal is more powerful than that.
    - Eemeli: Current <amount> would only reveal the formatted localized representation of the otherwise determined amount value in a tooltip, but not in the dom [missed]
    - Nic: String non-localized version of amount, browser generates nice localized version. Let's say we have a component that takes a JS Amount value, use Intl.NumberFormat to format it, use that as contents of html element.
    - Shane: Like idea of pointing to templating engines. Kind of the MessageFormat use case: take the amount and do the right thing with it. This is about the interop. People seem intrigued by this idea, but communicating it better is what we should try to do better.
    - Eemeli: Talking about this in w3c i18n wg. Nic + Eemeli will coordinate on breakout session about Amount and TG2 matters
- Shane: Summary of call:
    - Went over four points of feedback from v8
        - Avoid new numeric type
        - Name
            - FixedAmount, PlainAmount, AnnotatedNumeric
            - Try using Amount as plan A. Enough other options that we can agree that if we did have to move forward with Amount* as the conditional stage 2, we could still do that.

                - Futureproofing: “Annotation” as possible future-proofing
            - Protocol
            - Integration with html <amount>
                - Already engaged with champion, need to engage more
                - Emphasize idea that Amount is not just for html amount, but in general for any framework w/ templating engine
    - Still try for Stage 2 on Wednesday. V8 team is content with us asking for Stage 2 given these things are explicitly noted as requirements that we should be addressing in Stage 2.
- Nicolo: WH concerns
    - Two PRs up addressing significant digits issues.
    - Shane: WH at least supports the proposal, emphasize that things we can resolve in Stage 2.
    - Nicolo: Same situation with Decimal
- Eemeli: Are we aligned with the protocol being a String representation of the Amount?
    - Shane: Open to String being the protocol, don’t need to commit to it right now it’s a clean design.
- Nicolo: I like simplicity of String, if duck-typing, object though
- Jesse: might get killed by people worried about parsing, duck-typing is reasonable
- Shane: Slot, then fall back to String. Most of the time we don’t do parsing, I think the parsing is efficient because the string representation of the decimal is the form that Intl uses.
- Eemeli: syntax here is not amenable to addign additional property fields. If we want to rule out that that’s a direction we want to enable, then Strings are quite valid. There’s in particular if we end up with fractionDigits or significantDigits as a field as an object that related to formatting, then that’s problematic because the fractionDigits of Amount and Intl.NumberFormat mean different things. “Scale” better, since that’s not a term wt use as Intl.NumberFormat.
- Shane: String form could allow multiple annotations. In the same way that RFC 9557, time zone doesn’t need to be annotated. Unit gets annotated as default annotation, if we need error bars it would be e-eb=+-5, whatever syntax. We shouldn’t bring up string protocol because we’ll fall down rabbitholes. Focus on Stage 2 problems
- Eemeli: If we promise to consider names other than Amount – oh, should we strip out toBigInt and toNumber?
- Shane: Yes, it’s safer. V8 team was happy about the spec not having them. Personal opinion: we can still have them if we really work on designing their signatures well, prevent people think that that they roundtrip correctly.
- Eemeli: Fine with dropping toNumber and toBigInt, would prefer to do that in response to you presenting something that would ameliorate the V8 team’s position rather than us doing it proactively. Even if they’re not in the spec text, they are conceptually in the proposal. If we were to remove that, we should do so in response to a public statement so that we can get a feel for what the rest of the committee thinks of it. I wouldn’t be surprised if other members of the committee very much want to maintain toNumber/toBigInt

- Nicolo: Need to merge Jesse's bugfix. PR 63.
- Eemeli: So long as it doesn't *close* 54.
- Shane: I will run slides past the V8 team

2025-09-18

Attendees

- Ben Allen (BAN)
- Eemeli Aro (EAO)
- Nicolo Ribaudo (NRO)
- Jesse Alama (JMN)
- Richard Gibson (RGN)
- Shane Carr (SFC)

Agenda

- Any feedback wanted about upcoming plenary talks?
- Imputing additional precision in amounts
  https://github.com/tc39/proposal-amount/issues/49
    - Follow-on discussion: creating an Amount from an Amount

Next steps

- Try to get a resolution to issue #49
- Parsing numbers-with-unit in the constructor (we don't know what to do, present this as an open question to plenary)
- Check Intl.NF working with 3e-5 in keep-trailing-zeros
- Should it be possible for Amount to take an Amount as argument? Make new issue to discuss, link to https://github.com/tc39/proposal-amount/issues/49

Notes

Amount:

JMN: One approach I've had to thinking about the constructor is that there's always a separation from the numeric value and the options, including number.

EAO: I think we're saying we should have an issue filed about parsing string representations with units, include this as an open question for next week.

Decimal:

EAO: What are next steps?
JMN: Need to clean up spec text and open issues, Amount to Stage 2 unlocks Decimal, make sure it's ready for Stage 2 in November

EAO: Does the way to which Amount is enabling interchange changing some of the expectations to Decimal in a way that requires those to be refounded again?

JMN: It's true that Amount is taking up a little bit of the use cases for Decimal, some do get cannibalized by Amount. Amount to Stage 2, though, gives breathing space for Decimal. Things involving arithmetic are not present in Amount, that's one of the reasons for Decimal, but some of the interchange stuff gets taken into consideration by Amount.

SFC: I think that Amount solves the use cases that I've been trying to get into Decimal, now that Amount is a separate proposal then Decimal can focus on arithmetic / doing arithmetic in an ergonomic way.

EAO: Then, if you're going to do that, then it is valid to also ask if it continues to make sense for Decimal, if it's only there for arithmetic, to define its own Decimal object, or if it should be a set of methods on Math or something else that's capable of representing its values as Numbers or Amounts or string decimal values.

SFC: I've previously thought that you can do decimal math without having a Decimal object, but I hear JMN's argument that it's much more ergonomic to have a Decimal object. Might be question for committee. We could have it on Math, we could even say we're going to add as a separate proposal the arithmetic functions to the Amount prototype. That's a potential direction, more of an ergonomics question.

EAO: If we do that we're back to implementing BigDecimal, effectively. If Decimal is not adding a new Decimal but is working on other representations of numbers in a post-Amount world, then it becomes hard to rationalize having it limited to the number of bytes in Decimal at the moment. If the rationalization of Decimal is "we want to do Decimal arithmetic", then an upper limit of 128 bits is unnecessarily limiting.

JMN: Agree. I was never quite satisfied with Decimal128, reasoning was that BigDecimal has issues with division due to representing number of digits that division would have, some performance concerns about BigDecimal, concern for runaway calculations that produce large numbers of digits, rational has similar concerns, but then again the proposal *is* inconsistent with its description, since it does explicitly say that high-performance is a non-priority. How far is it *not* a priority, though? Do we need experimentation on memory usage? A long time ago MF and SFC sketched the idea of rational numbers, I think that's still on the table, it's not right next to me on the table, but it's there, it's not too different from what we're doing with BigDecimal, but in

a different form. I like the opportunity to delete some of the motivations / make it focused on arithmetic.

SFC: This is why I've been advocating for an all-in-one numerics proposal, since it seems wrong to think of these things one six months after the other. If Amount gets to Stage 2, we can even make tweaks to Amount based on needs of Decimal. I think both should get to Stage 2 before we promote either to Stage 3.

EAO: I would like to know – the current shaping of Amount to support all numeric-ish values in JavaScript, makes it much easier to think about adding support for Decimal, Rational, or other number-ish value in Amount. We're already in a state where you can construct an Amount from a BigInt that's not valid as a Number or construct an Amount from a Number that's not valid as a BigInt. Expanding that to support other numeric representation allows for that same thing. Do we want to special-case the Amount you get from building an Amount on Math.PI? Math.PI and Math.e are Numbers, right? But it would not take that much to build a value of pi, e, other constants we have to a higher precision than we have currently.

JMN: I did in early sketches for Decimal have Decimal.pi, Decimal.e, Decimal.log10 as Decimal values. We certainly can do that.

EAO: That would make Math.pi a little exotic, though, in order to detect them.

SFC: w/r/t this representation, we've gotten to a point, to a realization that there's not much of a difference between saying the inner type of Amount is a String vs. saying it's an Intl MV vs an enumeration over all JS numeric types are all basically ways of saying the same thing. Current path does resolve concerns from MM, WH. Given that we're in this situation, I agree with EAO that adding Rationals and other things is a much more reasonable step to take. Speaking of Rationals, since the Decimal proposal takes into account arithmetic, if we could do Decimal by adding arithmetic to Amount, then we could do the same with Rational, have a decimalDivide, rationalDivide, something like that.

EAO: If the operations are on Amount, *and* if a rational representation of numeric values exists in JS, then the Rational proposal is going to provide a way to go from a Rational to a Number, and then what Amount does can just rely on that.

EAO: Oh! We need to figure out precision. We have problems with precision when representing numbers as 10 to the power of something. What should be an easy question: how many significance digits do we get if we start from 3e - 5.

JMN: One, I think?

EAO: 3e - 5 formatted with one significant digit is zero. I think the thing is that significant digits needs two numbers, and one is the number of significant digits, and the second is what is the

the “e number”, the exponent, that this is applying to. I haven’t worked out the math on this, but it gets interesting.

JMN: I wonder what Decimal128 does
EAO: Or we need to say that the significant digits in 3e - 5 is 5, which feels wrong but gives the right formatted result out. Except if you do that then you’re expressing it again as an exponent, you’d get 3.0000e - 5. I don’t have a solution, but I think there’s something here that we’re not figuring out correctly.

JMN: So Amount is a number + a precision, that might lead us into difficulties – do you think there’s already a difficult in Amount, or are you imagining just Rational.

EAO: In Amount. IIRC in keep-trailing-zeros, that one accounts for the exponent when calculating precision. I propose that we allow the significant digits to be inspected from outside. If you construct an Amount from 3e - 5, you’d expect it to be 1, but the amount we should report is 5 (or perhaps 4)

JMN: So the power of ten that this sits in? That’s a property of the mv regardless of the notation. Or is it a problem with the notation? I agree that when we construct an Amount from a String in exponential notation, that’s a problem, you’re saying we do some calculation and then use the results to determine what power of ten to use?

EAO: I think there’s a difference in how we’re having significant digits show up in Amount and how Intl.NumberFormat handles significant digits. If you pass that into NumberFormat with significant digits: 1 you get 0 out. Because when Intl.NumberFormat is dealing with a Number it’s just an mv, it forgets the precision, and then it applies the precision afterwards.

JMN: Does this even interact with keep-trailing-zeros?

EAO: No, keep-trailing-zeros just adds the exponent digits, I don’t remember how exactly it went, I’ll have to check my own work to make sure it deals correctly with trailing zeros when using exponent string values. WH might have also filed an issue in Amount about some of this.

2025-09-04

Attendees

- Ben Allen (BAN)
- Eemeli Aro (EAO)
- Nicolo Ribaudo (NRO)
- Jesse Alama (JMN)

- Richard Gibson (RGN)
- Shane Carr (SFC)

Agenda

- Support for infinity in Amount as well as NaN and -0
- Strategy for next plenary in September
    - Amount for stage 2?
    - Decimal for stage 2? (unlikely)
    - Keep trailing zeros for stage 3?
    - Smart Units??
    - Any Number-related Intl updates? (e.g. PluralRules should support BigInt or mixed units)?

Notes

- JMN: Majority of people leaning toward including Infinity
- NRO: What about NaN and -0?
- JMN: Reasons for supporting NaN and -0 more or less the same. If we take the reason for accepting +/- Infinity, this extends to Nan/-0. Aligning with IEEE 753 and IntlMV
- EAO: Yes, agree on NaN and -0, what convinced me was argument that WH made, that this thing looks like it wraps Numbers. If it looks like it wraps Numbers, it must support all Numbers (and all BigInts). Because Infinity is a valid Number, therefore we must support Infinity. By same logic we must support -0, -Infinity, NaN. We should look at the existing thing we are supporting rather than making a domain.
- JMN: We're making a union of domains.
- EAO: Aspect of this: If we accept this argument it follows we must support all BigInts, meaning we can't assign an upper limit to what values of BigInts we accept. So how do we mesh this for desires for an upper limit in IntlMV? Do we potentially say that because Decimal string is a vague thing that we don't well-define, do we assign an upper limit to decimal string values but not BigInt values? We could do that, but it would be weird.
- JMN: Agree. We're walking on thin ice/closing our eyes to issues of limits, but I'd be happy to pretend at the spec level that we've got an unlimited domain, though as with BigInt there's going to be practical limits there. Maybe we could have a note?
- NRO: [something]
- JMN: Browsers give a RangeError when BigInts are too big
- NRO: Err toward being more lenient rather than crashing out.
- EAO: The spec doesn't set an upper limit for BigInts, so what do we do with BigInts bigger than supported by IntlMV? Do we accept it, do we round to infinity…
- JMN: Can we say NumberFormat is allowed to throw?
- NRO: NumberFormat already rounds to Infinity. NF does what Amount does in the draft spec and +/- Infinity, -0, and NaN. Rounded when outside that domain

- EAO: Intl.NumberFormat accepts all BigInts. Early return if the input is a BigInt
- NRO: Raises internal slots issue discussed in TG3
- JMN: That issue potentially solved in Amount
- EAO: I do not think we should special case / consider BigInt wrapper objects that are not BigInt primitives. My recollection is that the easy ways of doing so shouldn't be allowed
- NRO: Correct
- EAO: We should not allow for information on whether the thing was a BigInt to pass through. Behaviour should be the same
- RGN: It's worth noting that Intl.NumberFormat does not behave that way
- EAO: The IntlMV that NF creates is never allowed to be user accessible anywhere. You only get a localized string
- RGN: But if your input was a string rather than a BigInt, you get a different localized string.
- EAO: So which behaviour do we go with?
- NRO: If we're going to have implementation-defined limits, I'd prefer to have spec-defined limits. I don't think including all BigInts is worth it if we're going to have implementation-defined limits
- EAO: I pretty much agree, but we're currently in a situation where we have extant implementation-defined rather than spec-defined limits. At next plenary, we should state a preference for limits (same as Intl.NumberFormat?)
- SFC: I'm planning on putting a separate agenda item for discussing open PR on IntlMV, making slides on general problem where for these numeric types if we don't specify limits implementations choose their own limits, which is not good for anyone. In terms of what to do with BigInts, I'd like to decouple that discussion from IntlMV and Amount, hopefully it's not a stage 2 blocker figuring out how Amount is bound on BigInt
- EAO: SFC, do I recall that your change to the upper/minimum limits for IntlMV do not change the behaviour around BigInt when creating an IntlMV out of a BigInt?
- SFC: Only apply to strings, not proposing to change BigInt behaviour, don't intend to do that right now, just want to make a narrow change right now, come back in November
- EAO: What may work is having two different numerics topics on agenda, one for Amount, one for IntlMV. The question is should we have *three* – a separate discussion on how to handle BigInts, i.e. if we can agree on behaviour for IntlMV on strings, can we then have a discussion on having a limit on BigInts
- RGN: Lots of heterogeneity of implementations, some assume low resource-availability, some with high resource-availability. An additional possibility is that the spec defines bounds rather than limits, i.e. spec says "implementation must support at least a given threshold, can go beyond that."
- NRO: In practice implementations are always allowed to throw when looking at something too large. Even if we define a limit implementations are allowed to throw if out of memory. Meaning we have errors rather than rounding, which may be undesirable.
- SFC: Out of scope for this call, point about implementation limits is something we've tried to get opinions on from Moddable, they didn't have strong opinions. A discussion I want to have is it better to have upper/lower bounds or just upper, should they be the

same, etc. Agree with EAO on order of presentations, not sold on pushing on BigInt as well this meeting or in November instead

- NRO: right now BigInts are not converted to mathematical values?
- SFC: BigInt converted to mv, strings parsed to mv, BigInts aren't limit-checked, strings are
- EAO: Solution that might solve most of the problems: define an explicit limit in Annex B but not the spec otherwise
- NRO: We're trying to move away from Annex B, active work to move it to the main spec
- EAO: But in this case where we have, effectively, an implementation-defined upper limit and don't want to expand the scope of it, having different behaviour for browsers compared to other implementations kind of sounds like this is what we actually want, the uncomfortable part is trying to make two different enough systems work the same way when they don't need to – they already don't in terms of BigInt. With the high limits we're raising IntlMV to, this debate seems largely academic – not invested in any solution so long as it makes some kind of sense
- NRO: w/r/t stage 2 blockers, we should just structure the discussion in terms of "here's a solution, if it doesn't work let's go ahead with this alternative"
- EAO: Related part that we should resolve before asking for Stage 2: as we are going beyond finite numbers, what are the helper methods we want to attach to Amount? We should have at least isFinite() and isNaN() – are we agreed that these should be on the Amount prototype? And is there a desire to add isInteger and other methods beyond this?
- JMN: Decimal currently has isNegative, only reason for including that is -0
- NRO: For isInteger, a workaround is checking is fractional digits 0.
- NRO: How difficult is it to check for NaN? Cast it to Number, if it's NaN then it's NaN? Or does that not work. I think it's safe for NaN, because nothing will cast to NaN if it's not NaN. Not the same case for infinity
- SFC: I support having all the helper methods. If too many the question becomes are there too many plug points for v8, but here we're talking about the difference between three and five functions
- NRO: Agree that we should have these
- EAO: Are all okay with isFinite and isNaN, or should we explicitly include isInteger or other methods?
- SFC: isInteger easily derived from fractiondigits: 0, interesting question to resolve though. Not ready to answer that question

Next steps

- For plenary: 3 agenda items: discussion of limits (esp. BigInt, but more generally); Amount; …

2025-08-21

Attendees

- Jesse Alama (JMN)
- Richard Gibson (RGN)

Agenda

- Support for infinity in Amount as well as NaN and -0
- Strategy for next plenary in September
    - Amount for stage 2?
    - Decimal for stage 2?
    - Keep trailing zeros for stage 3?
    - Smart Units??
    - Any Number-related Intl updates? (e.g. PluralRules should support BigInt or mixed units)?

Notes

No discussion, too few attendees.

Next steps

- Deadline for materials for September 2025 plenary is only 3 weeks away (September 12th)!

2025-07-10

Attendees

- Jesse Alama (JMN)
- Richard Gibson (RGN)
- Shane Carr (SFC)
- Nicolò Ribaudo (NRO)

Agenda

- Amounts and `toString`: always emit the unit? (Shane's suggestion for a displayUnit option, with e.g. `always`, `never`, `auto` as possible values, `auto` being the default)

- Constructor for Amounts: accept `roundingMode`?
- Wrapping up the Measure README updates (done)
- Wrap up spec text for Amount
- Prepare slides for Amount for stage 2

Notes

SFC: Mixed units work should be ready for the next TG2 meeting

- *Amounts and `toString`: always emit the unit? (Shane's suggestion for a displayUnit option, with e.g. `always`, `never`, `auto` as possible values, `auto` being the default)*
    - JMN: If I understand correctly, Eemeli is on board with Shane's suggestion. Or at least, he approved the PR. He'd hopefully lead to the topic without explicitly being happy with it.
    - EAO: A bit concerned about it being a numeric string with no units, and then it changes when you add a unit.
    - SFC: Similar concern, maybe the default should be "always"? Otoh, there is precedent in [...] to only add bracketed stuff if they are actually present.
    - EAO: Do we have a parser that support datetime timestamp without brackets but now with brackets?
    - SFC: Temporal parser parse the string and ignore the parts they don't need. If you take a fully qualified string with annotations and pass it to PlainDate.from, it will only consider what it needs. It will still to validate the whole string. Need to double-check.
    - EAO: Yeah, but Number(amountString) would return NaN
    - NRO: How do we convert an amount to number?
    - SFC: We would not go through a string. Either brand checking in Number constructor, or a toNumber method
    - EAO: My sense was that not adding brand checking would be cleaner. toNumber is cleaner than brand checks. Are we setting precedent? Are we following precedent?
    - RGN: Already temporal, so we are not setting precedent.
    - NRO: Temporal uses a .epochNanoseconds property, not a method
    - JMN: We should be explicit about conversions.
    - NRO: MM might block on brand checks in the constructor.
    - SFC: Number constructor already calls Symbol.toPrimitive. We could implement that for Amount. If we implemented that, that's a good option. Symbol.toNumber or Symbol.toDecimal.
    - NRO: plus also calls toPrimitive. Do we want amount + amount to throw?
    - RGN: Symbol.toPrimivei also has a hint: do you want a string or number?
    - NRO: MAybe not for addition, but for multiplication. ToPrimitive doesn't distinguish.
    - SFC: From Number constructor, number hint is passed. ToPrimitive can throw.

    - RGN: no, there's always a hint.
    - SFC: If hint is "default", …
    - EAO: Let's avoid 4 euros + 5 kilograms equals number 9.
    - NRO: a++ passes number hint. All binary operators pass "default".
    - EAO: I'm ok throwing on default, handling number and string corectly.
    - NRO: OK then 1 kg ++ would yield 2.
    - RGN: 9 kg < 10 micrograms. Feels bad.
    - EAO: Number toDecimal…extend parser for Number to understand bracket syntax?
    - EAO: have .toNumber, but in ToPrimitive, be OK with giving out a Number value if we don't have a unit/currency. If those are present, then throw.
    - SFC: yeah.
    - NRO: sounds reasonable.
    - SFC: compare function?
    - JMN: sure
    - SFC: stage 2 concern.
- *Constructor for Amounts: accept `roundingMode`?*
    - JMN: I was thinking about just using .with for fracitonalDigits/significantDigits, but Eemeli convinced me that it's fine to have it there. What about rounding?
    - EAO: The cleanest way to understand it is to have the same options everywhere. And yes, .with would complain in certain cases, but with the same signature.
    - SFC: I thought we'd (maybe) round in the constructor, to avoid multiple rounding. We should decide this before stage 2.
    - EAO: difference between what I'm saying and what SFC is saying. Cleaner approach is to round only on the way out. Never do double rounding. We could raise this in the wider committee. We may want to retain the rounding mode to convert to decimal or number. LEt's say we have greater precision than decimal, and we may need to do rounding. If user has explicitly specified it, then we should account for it.
    - NRO: I think like SFC. I thought we round going in. Makes more sense to me that we do the rounding where it's needed.
    - SFC: New language rounding on the way in vs. out. I haven't heard that, let's use those terms. We should use this terminology. LEt's have a slide in the presentation.
    - NRO: PAss rounding mode to toDecimal and toNumber functions. Number is already rounded, but we could round again. We would specify, at conversion, how rounding should be done.
    - EAO: MAthematical value + bag of options mental model. Like Intl.NumberFormat. This is a discussion we need to raise. KEep the door open for both approaches.
    - SFC: Rounding on the way out: toString should roundtrip, but toString will emit a unit in brackets.
    - EAO: That would be cleaner. Retain parsed representation of the number. It's easier to have just one thing in the brackets. Uppercase, lowercase, digit 1.

    - JMN: should we parse these things going on?
    - NRO:
- EAO: Do we want to change the NumberFormat options that would throw if unit/currency is not provided to now throw?
    - JMN: Now, probably?
    - EAO: Because that feels like one of the places where we're changing current behavior.
    - NRO: I'd be surprised if that breaks anybody.
    - EAO: I think this should already be in the spec text for stage 2
    - NRO/EAO: At least having a note in the spec saying we plan to change it
- JMN: Ready for stage 2?
    - EAO: Open discussion points are @@toPrimitive, toNumber/toDecimal
    - SFC: Let's just decide on them, and try for stage 2
    - JMN: Igalia makes slides, Eemili presents?
    - SFC: I'd prefer to have rounding-on-the-way-in in the proposal, but keep it open for after stage 2 if plenary lets us.
    - NRO: agree we should have rounding on the way in. We can present, but we should have an option. Bring it up. If strong opinions, we can go to stage 2 with whatever the outcome of that discussion is.
    - EAO: There will always be rounding on the way out. toNumber and toDecimal have known limits. We would round on the way in AND out. So the discussion isn't in or out, it's both or only out.
    - SFC: if rounding on the way in, intl.numberformat should be updated to not do rounding on the way out. Even specifying maxFraction digits etc., we may need to round.
    - NRO: What about 1.46?
    - SFC: Won't happen, we have enough precision.
- EAO: keep trailing zeros:
    - Updated spec PR. Mostly following JMN's and SFC's comments. I'll package that into something presentable for the 402 call, hopefully get 2 or even 2.7.
    - NRO: I can review for 2.7.
    - SFC: I want to review it, too.
    - EAO: I can ask 2.7 for that.
    - NRO: Directly asking for stage 2.7 might be conditional. SFC was already mentioned as a reviewer. I have to say that I've volunteered outside of plenary, check that with committee.
    - EAO: Just setting up for 2.7. If that doesn't work, 2.7 should be easy.
- RGN: Not feeling good about the way currency is privileged.

Next steps

- Prepare slides for Measure for stage 2
- Allow "1.2[kg]" in the constructor (currently not allowed)
- Change Intl.NmberFormat to accept merely style: unit/currently without throwing.

- Add toNumber to spec text
- ToPrimitive: should give number or string if no currency/unit, throw otherwise.
- RGN updates us about concerns re: currency in the chat or as an issue

2025-07-02

Attendees

- Jesse Alama (JMN)
- Eemeli Aro (EAO)
- Richard Gibson (RGN)
- Shane Carr (SFC)
- Your Name Here (XXX)

Agenda

- Spec text for proposal-keep-trailing-zeroes
- Updated README for proposal-measure (reduced scope to focus on Amount, with unit/currency but no unit conversion)

Notes

Next steps

- Review Eemeli's PR
- Consider removing rounding in Amount constructor
- Define how NumberFormat should work with Amounts (currently only PluralRules behavior is present in the draft spec text)

2025-06-18

Attendees

- Jesse Alama (JMN)
- Eemeli Aro (EAO)
- Shane Carr (SFC)
- Nicolò Ribaudo (NRO)
- Your Name Here (XXX)

Agenda

- Next plenary plans. Jesse had posted in Matrix:
    - for July plenary: spec text and tests for proposal-intl-keep-trailing-zeros, going to stage 2 or even 2.7
    - measure proposal renamed to "amount", scope gets reduced, data model changes from decimals to digit strings
    - for July plenary: amount proposal for stage 2
    - amount removed from decimal proposal (because that functionality moves to proposal-amount), spec text polished and completed
    - for July plenary: no decimal update, no stage advancement
    - for September plenary: ask for stage 2 for decimal
    - smart units remains on the radar and up for discussion, though currently lacks a plan and schedule
-

Notes

- Trailing zeroes for stage 2 in July, still missing spec text
- For Measure, make sure that its readme is up-to-date:
    - Present Amount
    - Make sure that motivation is there
    - Mention what's related but we don't want to advance as amount but move to smart units
- Move unit conversion to smart units. It's slow because of two problems:
    - How to have user preferences in the web platform
    - Conversion, with an API that does not lead to abuse
- Measure:
    - Representation of unit/digits/value, nothing else
- Another spinoff from measure: foot-and-inch compound units. TG2 proposal needed? Measure should be agnostic about that.
    - Shane: Whatever Intl.NumberFormat supports is supported by measure.
    - Eemeli: foot-and-inch is in stage one, removing it would maybe communicate incorrectly about what we want?
- Jesse: what’s the status of the amount of discussions in Mozilla? Do we need to work with the web folks?
    - Eemeli: the "web folks at Mozilla" is tantek, but he's not been pushing Amount ahead recently. Time element? Exists, doesn’t do anything. If we can write stuff into the DOM, can we change how <time> works to do DateTime formatting, etc. And that could be a smaller precursor to something like <amount>.
    - Nic: some sort of declarative way for Temporal+DateTimeFormat or Amount+NumberFormat?

    - Eemeli: yes, kind of. Also useful for server side rendering that supports user-localized content. Eemeli can talk to Tantek.
    - Shane: HTML could be a side note that we mention in the amount proposal, but it's best to frame it as motivated by the needs of *JavaScript* devs.
- Nic: rethink the question: Why global amount, not Intl.Amount, especially if we do a string-based amount?
    - Shane: Numerics.Amount, Numerics.Decimal (namespaced). We do have issues (e.g. 175) that this isn't just Intl stuff. Useful for any API that processes numbers with dimensions, more common than bare numbers. Immutable interface for interoperability. That's the main thing, and if it's not enough mention the full decimal data model + financial.
    - Eemeli: What Shane said needs to be explicitly noted in the readme, mentioning also HTML.
    - Eemeli: Another thing is that because we can point out some non-Intl use cases for amount, placing it in Intl is saying "Intl is not just about i18n anymore",
    - Shane: this is a good point. Even small use cases shouldn't require 402.
    - Eemeli: let's not encourage using Intl for "whatever".
- Eemeli: I'll present a proposal for foot-and-inch by itself. Technically it's already as stage 1 in measure.
    - Shane: I'd still present it as seeking stage 1.
- Shane: Eemili, can you make sure that your colleagues are happy with amount?
    - Eemili: I'll need the materials first.
- Namespacing: offer it as a possibility, but low-key.

Next steps

- README update for amount/measure. Will put that work in the exiting spec text PR.
- Eemeli will write spec text for trailing zeroes.
- Eemeli can't do a TG2 proposal for foot-and-inch (compound units). Shane can perhaps get that ready for stage 1 in July.

2025-05-08

Attendees

- Jesse Alama (JMN)
- Eemeli Aro (EAO)
- Shane Carr (SFC)
- Richard Gibson (RGN)
- Your Name Here (XXX)

Agenda

- EAO's proposal to make Intl.NumberFormat.p.format produce trailing zeros if given a digit string containing them

Notes

[EAO presents the README of the repo, as well as issue 1]

NRO: What does web incompatibility look like?

EAO: this should be rare

NRO: What about other options?

EAO: current behavior continues exactly. Don't foresee a problem.

JMN: Yes, this should be rare. A couple reasons to think that, both for NF and PluralRules.

NRO: what's the process for 402?

EAO: We present it, agree (possibly after a lot of discussion) to go to TG1. They will probably also have opinions.

NRO: One concern in TG1: speed.

EAO: We need to do toString anyway. Whether we do it once or twice is an implementation-internal detail.

NRO: In Intl.NF.p.format, if I give you a number, do you call toString on it?

EAO: We need a string representation, equivalent to toString, and work with that.

NRO: Thinking about the details: we need to upgrade the definition of intl mathematical value. Should be a struct.

EAO: Yes.

NRO: This might make things tricky with Decimal.Amount.

EAO: We have three kinds of numbers: Number, BigInt, and digit strings.

RGN: Is that right? Isn't the digit string just Intl?

EAO: …

RGN: Grammar in 262 doesn't correspond got 402. Wider syntax for numbers (octal, whitespace, etc.)

NRO: [shows the spec text]

RGN: A number of edge cases to consider.

NRO: If we do this, should we be able to query the string?

RGN: Exposed?

NRO: Yeah.

RGN: Where to expose it? Not a method on String, I think. Weird as a method on Number.

RGN: Could be a static function on String. Probably where it would end up. Convert input to a String. Require it parse with the grammar, but not convert it to a Number.

EAO: I'd need to see use cases.

RGN: Right. I want to see where it goes.

EAO: I'd like to see it on Number.

RGN: That's bad, because Number doesn't work that way.

JMN: What about Decimal.Amount? I didn't mention String.Amount in the issue.
EAO: Having a toplevel Amount that could do the job is not the worst thing we could do. NF is happy with any number up to MAX_NUMBER_VALUE. You can get up to 307-308 integer digits, up to 100 fractional digits.

RGN: Range of all decimal representations with no more than 100 fractional digits, which converted to number, would be finite.

EAO: I support that, even if there accidentally. No need to format numbers bigger than that.

RGN: What about "1e10000"? Is it specified?

EAO: Could be related to ICU4C internals.

RGN: I'd liek the specification to include that limit. Don't care what the limit is.

EAO: As a consequence of this, SFC has mentioned using decimal as an internal representation of number being formatted. But we currently support using numbers way outside the range of Decimal128. Limits on fractional digits were raised from 20 to 100, and that was enough.

SFC: MM had use cases for decimal that didn't fit. What were those?

RGN: Crypto.

EAO: I'd say we have a limit to about 400 significant digits. Going down to 10% of that would probably annoy some users.

RGN: Maybe we need a stress test. Very precisely measured physical constants. Fine structure constant.

EAO: Silly number of digits might come up. These aren't necessarily significant use cases. There may be use cases out there, though, with huge limits.

—--

JMN: [presents idea of doing Number.Amount, maybe BigInt.Amount?]

NRO: If we do this,maybe we should have a string amount.

EAO: We need to have a reason for what we're doing. IF we have amount wrapping around a value, what should it be? In my view, we should have strings. We need SFC to talk about non-Intl use cases. This discussion needs to happen again with SFC.

RGN: Roundtripping should work for digit strings, possibly excluding -0.

NRO: What about toPrecision?

RGN: The relevant part of the spec is canonical numeric string. The intention is for this to be roundtrippable.

EAO: An amount could give its value only as a string.

NRO: What if we don't know how the value was created?

EAO: Why is that a problem? TypeScript type for this should be type of the argument. But in JS this info would get lost.

NRO: Relates to the safe number proposal, which didn't get to stage 1.

EAO: You can construct and consume amounts around number, and it would work. Amounts created around decimal. But mix and match would be like creating a number from a decimal or vice versa.

NRO: What about mixing amounts created in two different ways?

EAO: Is that really going to happen? It feels that this is at the boundary of JS: creating them, and then passing them out.

NRO: think about equations in math, comparing numbers; versus equations in physics, with units.

EAO: This sounds safer than using raw decimal values. Given what these amounts are of, unless dealing with crypto, they should fit well within the part of number that decimal correctly represents.

JMN: We have an equals method.

NRO: if units are normalized.

JMN: IIRC SFC wanted equality to not necessarily canonicalize, similar to Temporal's comparisons.

2025-04-24

Attendees

- Jesse Alama (JMN)
- Eemeli Aro (EAO)
- Shane Carr (SFC)
- Richard Gibson (RGN)

Agenda

- Next steps after last plenary:
    - Intl allowing bare decimals

- Is the “amount” idea a good candidate for being a protocol?
- Should a decimal string be our “amount”?

Notes

EAO: If we want to fix Intl to handle trailing zeroes, should that be a separate proposal? If you look at 402 and parsing of decimal strings, this might motivate the idea. Maybe RGN knows.

RGN: Not sure, need to think about it.

EAO: In Intl we have Intl mathematical values. Not sure how to implement that in the right way.

RGN: Intl mathematical value can’t handle trailing zeroes directly. 1.200 is 1.2.

EAO: We would need to keep that data somewhere, during parsing. Maybe we could grow the definition.

RGN: It would sort of grow to a tuple.

EAO: That kind of change could be presented separately from decimal. It could be a completely 402 proposal. Decimal could presume this functionality exists. If no use cases outside of Intl can be found, then this is more motivation.

JMN: What about decimal killing any trailing zeroes during parsing?

EAO: Not aware of anything right now that preserves trailing zeroes.
JMN: summarizes the rough idea of changing the definition of Intl mathematical value

SFC: but that doesn’t change any APIs

EAO:

SFC: TYou (EAO) mentioned making string the intermediate representation. But we don’t have agreement about that.

EAO: Right. So I’m talking about a separate proposal, where strings with trailing zeroes get handled separately.

SFC: Thoughts about plenary still need to be written up. No one pushed back about the idea of a new class. The only feedback was from MM, who said that this should work for Number, too. The Decimal.Amount is that intermediate type. It works with Number because it handles all numbers. We got some pushback about WH too, about fractional digits vs. trailing zeroes. I’m unclear why we’re stuck on this. Part of this might be a lack of clarity among the delegates about the data model. The intermediate type doesn’t have memory of how it was constructed.

That's what EAO is ujp to when he talks about strings as an intermediate data model. It's not just a string but an upgraded data type.

EAO: My assumptions walking into plenary: something like Decimal.Amount would retain the full decimal value. So it would have a "significant digits" field. So yes, you (SFC) predicted my thoughts. So if it's opaque and we apply fraction/significant digits when we create it, we get something that, for the use case presented in the decimal proposal, I'm not convinced. Provided that Intl can't preserve that info.

SFC: That's valid. If this logic…maybe we didn't present this clearly. If we here in this call didn't know, MM and WH probably didn't know. If we communicate with them better that this intermediate intrinsic represents everything that a number string can handle, but is backed by a decimal.

EAO: MM didn't like that it's backed by a decimal.because of the upper limit. Bitcoin use cases.

SFC: He dropped the bitcoin use case. He said "if it works for decimal it should work for number".

EAO: I agree that we can go that route.

JMN: I'm not sure about that. It's my job to provide a counterexample.

SFC: I've presented a definition of number. If that works for our case, we can do it.

EAO: If we have any use cases outside of Intl, …
SFC: Yes. There's demand for having a built-in for a number with unit. There's definite demand for that. 2nd use case is the ability to parse and round-trip numbers from other platforms. If we have a "Decimal128" class, it seems bad to not interoperate. Ruby is the only one I found that doesn't do this. Other languages have precision. So if we want to interoperate with them, we should be able to parse and round-trip such values. List is in issue 175. A third use case: financial calculations do care about precision of values. Having these issues is well-motivated outside of Intl. Corollary: Intl is the primary motivation. But the argument remains. Still not the only motivation. Calling it "Intl.Amount" isn't the right way to go. I don't even necessarily agree that Intl is the only motivation. Should be in 262.

EAO: For the 2nd use case, interoperability: what is lacking with its own toString?

RGN: Can you describe the nature of round-tripping?

SFC: 2 instances: Talking with another language over a protocol e.g. a RESTful interface talking to a server, reading values. Other one: use a library implemented in Wasm, we want to interoperate. If you compile these into Wasm, these values might show up. Wasmgc. We don't want these operations to be lossy.

RGN: IIUC EAO was talking about a string representation as a simple example. We need a wire format. Interchange requires some representation. Binary, a string that might have trailing zeroes…you can have that regardless of the API within JS.

SFC: If you get a string intermediate and say “this represent into a Decimal128”, you can lose the trailing zeroes. The industry has settled on this. To interoperate you need the trailing zeroes. My position: the decimal proposal is adding a new numeric type for JS, solving footguns…this is not what the ecosystem has done. We’re being a trailblazer here. If it wants to interoperate, then trailing zeroes are part of the picture.

EAO: Not quite understanding. Let’s imagine we have an external data source, with trailing zeroes. “1.20” comes in. How do we get the precision out of this?

SFC: Let’s say we have decimal.amount. You get .from and you have “1.200” in its internal representation.

EAO: But the current proposal for decimal.amount, we get a JS decimal…

SFC: Not sure what you mean. We get a decimal amount that contains this info.

JMN: We have a separate constructor for this.

EAO: Would we have an accessor?

SFC: Accessor for what?

EAO: Accessor for precision. Let’s say I have “1.20” and I want to double this value. How do I do that?

RGN: The general answer is, you don’t and you can’t. Doubling can change the amount of significant digits. Developers would need to understand the intended correct output. Doubling isn’t well-defined regarding mutation of precision.

SFC: [presents example]

EAO: I understand this use case. Not yet convinced by it. You mention that other systems have a decimal that retains precision. So presumably they have a way of round-tripping.

SFC: I think I have some examples in one of the issues.

RGN: This example is nice because it’s comprehensible and unsurprising.

EAO: Is Decimal.Amount a strictly better choice than a numeric string? If we posit numberformat and pluralrules handling such values correctly, I think we need to show the need.

SFC: Having the functions that compute the significant vs. fractional digits.

JMN: Right, we can support that with convenience functions or calculation at construction time.

EAO: I'm trying to find the easiest solution. We need to show that Decimal.Amount satisfies that. We have strings as numbers in many places in Intl.

SFC: One of the compelling things behind decimal.amount is that we can add a unit field. Which is what you have wanted. From a non-technical point of view, you're sort of arguing against yourself. If we do this, it makes measure easy to move forward.

EAO: My main interest is looking at ways is not making the language/core library more complex to get us what we want out of it. If the cost is that we need unit and currency handling supporting its own weight, or by having a protocol, then that's kind of not horrible.

RGN: I think your position isn't incoherent. I think your concern about "do we need this" is a good one. Others in committee also have this position.

EAO: We need to be able to handle, in the decimal proposal. Rationalize why this amount thing needs to exist. Without referring to measure.

SFC: I'm looking at what the language looks like in a year, with both proposals advanced. I don't want us to be shipping proposals. I want a cohesive standard lib. We are able to build a more cohesive whole if we think of them together than in isolate. Let's suppose we have strings as the decimal intermediate for precision, then we have a protocol for units and currencies. That outcome is less good than if we design these together. That's why I think the proposals should be coupled. Sound't be shipping proposals as such.

EAO: I don't disagree. With the current split, handling precision handling in the decimal string, we need to argue that strings aren't enough. What was your third use case?

SFC: Ecosystem demand for having types that associate decimals with precision. 2nd was round-trippability. 3rd example was financial calculations.

EAO: How is that different from the round-tripping?

SFC: We have calculations.

RGN: Another example: when precision is lower than the scale. Representing that as a numeric string alone is impossible. "1.2" times "10^3". You can't know unless you have some other place to look—"1200" can have two or three or four significant digits.

EAO: Challenging for parsing as well. E.g., “1.2e3”. What I get out of this is that the use cases for decimal.amount are (also) non-intl. If so, it’s more clearly part of decimal than the amount proposal. Those use cases are better reflected.

2025-04-10

Attendees

- Jesse Alama (JMN)
- Your Name Here (XXX)
- Your Name Here (XXX)
- Your Name Here (XXX)

Agenda

- Feedback from DE and EAO about `Decimal.Amount`

Notes

Do we want this to be in the measure proposal or decimal? Sentiment was for this to be in measure, with a range from weak to strong support. An argument could be made for including it in decimal but not a strong one.

2025-03-27

Attendees

- Jesse Alama (JMN)
- Your Name Here (XXX)
- Your Name Here (XXX)
- Your Name Here (XXX)

Agenda

- Banning bare decimal objects in Intl formatters?

- https://github.com/tc39/proposal-measure/issues/26

Notes

Banning bare decimal objects in Intl formatters?

NRO: never pass a decimal to NumberFormat or PluralRules. Or: ban unless a precision is specified?

JMN: e.g., specify precision/digits in advance? As with toString.

NRO: yeah.

SFC: two conflicting concerns: adding a new numeric type should prompt us to get I10n right. Ability to localize a decimal directly is an onramp. What's correct? First convert to amount before proceeding. Or, specify digit options specified, as NRO suggested. Another is NF works but pluralrules doesn't. Or be strict: doesn't work at all, has to be converted to an amount. Or be lenient. Not sure what committee would like.

NRO: most people in plenary don't have strong designs about Intl APIs. Prefer simplicity: always throw, never throw; easier to explain and implement.

JMN: I always thought about accepting decimals.

NRO: if there were no amount in the picture, we should accept decimal.s But with amount, we could do a strict version (no decimals, but amounts).
SFC: I suggest we support it in NF and not PluralRules.

NRO: What about NF and PluralRules? Anything else?

SFC: Those are the two places for Intl mathematical value. There are other places where we accept JS numbers. They're more strict in what we accept. Decimals probably don't go there.

NRO: I don't dislike the idea. What about updating the spec text for that? Add a note explaining that precision needs to be specified.

JMN: Make an issue?

NRO: there is an issue.

SFC: Once we have more cookbook examples. Last week, we talked about having a cookbook. These older issues can be turned into examples.

JMN: I can ping Eemeli about this.

https://github.com/tc39/proposal-measure/issues/26

SFC: (presents the issue) NB: this is immutable, this may be important for some delegates. Emphasize ergonmicness, too.

NRO: what about “max” fractional digits?

SFC: sorry that’s a mistake. PRobabnly we should have min/max being the same/

NRO: that’s what I wanted to say to MM in plenary. This can clarify the possibility of changing.

SFC: Looking at JMN’s comments. Yes, we could normalize units. Maybe in a future proposal. But we could do this in v1. Maybe just for SI units. This could give us a minimal version. To do a kilogram/gram conversion, maybe we could have a “normalizeSI” function, and then do equals. I keep the kg/g as different. Same in Temporal. 120 seconds != 2 minutes (not even including leap seconds). With balancing, you can first balance and then do a comparison.

NRO: So unit would be a string? And perhaps throw if we get a weird unit?

SFC: Not sure yet, tbd. Initially, we allow arbitrary things in the unit slot. If you use a convert/balance function, then you throw if the unit is weird.

SFC: JMN’s 2nd question. I tried to explain this earlier.

NRO: 2.20 versus 2.2 with 2 fractional digits.
SFC: No math on these. HAving math on things that track precision is too contentious. Math is done on decimals.

SFC: 3rd question. This is a decimal-backed amount. This relates to discussions like “should we have a bigint amount”? We can do that. Having this in the namespace makes it clear that it’s a core feature of this proposal. This can help us deal with earlier discussions. With Decimal.Amount, it’s clear what it is.

NRO: Maybe highlight the bigint amount example. People were asking “wjhy in JS instead of Intl”. Being able to expand this to BigInt makes it more attractive.

NRO: I like this. Should we present this in plenary?

JMN: I’d like to chat with Eemli first before committing to that.

SFC: I’d like to have a presentation. “We heard your feedback in Seattle. We think this satisfies your feedback.” I think Decimal.Amount should be part of the decimal proposal. If we want to

have a Number.Amount, we can. Its semantics are similar. In terms of the concern that amojunts shold advance, then we can advance Number.Amount. Maybe we won't even need to advance it.

NRO: We can present this as an evolution of what we presented in JApan.

SFC: Right. I see this as a core part of the decimal proposal.

NRO: If we stil get pushback, we could slice this by keeping just the precision part and dump the unit part.

NRO: Do we want spec text?

JMN: sgtm

SFC: Updated my issue with more details.

2025-02-13

Attendees

- Jesse Alama (JMN)
- Mikhail Barash (MBH)
- Shane Carr (SFC)
- Nicolò Ribaudo (NRO)
- Eemeli Aro (EAO)
- Your Name (XXX)

Agenda

- Welcome. Purpose and scope of recurring call
- Architecture. Polymorphic vs. non-polymorphic amounts/measures.
- Should measure/amount go beyond CLDR's list of units and supported operations?
- Do we want to version any of this (adding features in versions) or do we want to try to get it all in v1?
- Do we want to support custom units in measure/amount?
- Should currency amounts be handled in any special way (https://en.wikipedia.org/wiki/ISO_4217) or is it just another unit?
- Should we support arbitrary custom units (essentially, strings with no interpretation)?
- Your items here

Notes

- Shane: We should as a priority find a group position, and let the rest of the committee speak rather than just speak between us in plenary.
- One of the main topics is the architecture issue.
- Eemeli: what are the problems we are solving? Layered build-up of the overall solution should be kept open.
    - 1st step: opaque Amount. Constructor takes in strings, bigints, numbers. Enforces a subset of what decimal would allow. Advantage: wouldn't depend on decimal, could be passed in to Intl.NumberFormat. Later, see whether decimal's needs are addressed by this.
    - Decimal would build on top of amount. Concerns addressed by opaque amounts differ from those addressed by decimal.
- Nic: process: separate proposals, amount shouldn't be blocked by decimal, since amount, in this approach, is quite simple. So why merge proposals? The thinking was the opposite fear: decimal wouldn't go ahead because this opaque amount doesn't fit in. But if we can find a small piece that can move ahead, let's do it. Similar to how we work in the modules space today.
- Nic: about the grand vision: dislike the idea of a single class that does everything. Some complexity with units when we start to do math. We're basically talking about rational numbers here, with its normalization approach. Restricting this kind of math to unitless values simplifies things. For unitless values, the "unit" is just some extra baggage. When it comes to precision, the way it's done in IEEE 754 is usually not what people need, and anyway, there are multiple valid ways to do that. So perhaps not build in anything there.
- Eemeli: did you see my reply?
- Nic: we support rationals, sort of, with units.
- Eemeli: Intl does nothing with string values. There's a "-per-" infix that's supported.
- Nic: you support a simple fraction?
- Eemeli: yes. There may be cases where this kind of math might be formattable.
- Nic: I wasn't thinking so much of formatting but comparing numeric values.
- Shane: I see why we might want to advance parts of this, but procedurally, … Temporal has taken a long time, but that's OK because we've taken time to think about the pieces. If we had advanced temporal instances first, and then did other things later, this may have had some downstream impact. So I have a preference for having a full vision rather than working piecewise. Sum of the parts is greater than the parts. We need to solve this problem for Intl, for sure, but the case is stronger if seen as a part of vision.
- Eemeli: Shane, do you support advancing an opaque amount before we solve all the issues? Given that the opaque amount has such a small footprint. We can design this in such a way that we don't restrict ourselves later. Or do you imagine that we can't do this ahead of time, and we have to solve it all?
- Shane: I don't oppose a very restricted an amount proposal. Preference for a unified proposal, but won't block a sufficiently restricted amount proposal.
- Eemeli: I'm thinking of this in terms of impact of further dependencies on this whole stack. Intl.MessageFormat depends on the opaque amount part of this work, but doesn't

depend on the further parts. I think an opaque amount would advance more quickly than decimal and unit conversions. I'd rather not have Intl.MF waiting for the whole stack.

- Jesse: what about module harmony?
- Nic: the people working here have a grand vision. We spend a lot of time discussing the grand vision. Recommend having that written down somewhere rather than always discussing the big vision. We have a few proposals and haven't tried to merge. When we bring up big chunks to move, it's hard for committee to understand. Smaller pieces are easier to understand. But we've asked 2-3 times to present our grand vision. Feedback: people still struggle. Given the size of the vision here, we should make sure that the committee has the whole vision. We don't need smart units.
- Nic: if we want to have multiple proposals, what should they be? Should we cut things up as they are today, or redraw the lines? Might be helpful for committee. Order the stack. Tell the committee that we don't know yet how to do it.
- Eemeli: 1. Opaque measure/amount, 2. Decimal parts, 3. Conversion. Possibly in paralle with (3), extending the set of allowed units in Intl.NumberFormat (maybe a separate proposal).
- Shane: if we have decimal and amount advance together, then amount could be constructed from decimal. With amount alone, we need to think about constructor if the arg isn't a decimal. Decimal needs to answer these questions, too. If decimal were to exist, then measure/amount wouldn't need to solve this prorlbme.
- Nic: if we want measure to accept many things, then we need to restrict what the constructor can accept. So e.g. numbers might not roundtrip. IF we want amount to be explained as polymorphic, or, to make it forward compatible, we would need to restrict it.
- Eemeli: the opaque amount (see gist link), does not allow input value to come out except via toString. So this should be straightforward that will act the same as a later one that has an underlying decimal value. Constructor validates input, then passed to Intl.NF.
- Shane: re: opaqueness: an opaque toString throws away the unit and just presents the number part?
- Eemeli: yeah
- Shane: we should probably preserve some roundtripping here. As long as we restrict things to toString, this should be OK. But let's imagine we want amounts to adopt the polymorphic behavior. But then toString might need to change? But let's imagine a string decimal thathas 50 (?)_ significant digits. We'd need to cut that down. But if we want to make it polymorphic, maybe we don't want to do that kind of rounding. So we'd need to nail down Decimal's constructor before nailing down amount.
- Eemeli: for numeric string inputs that contain more digits than decimal can handle, we should throw in the constructor. That way, we preserve forward compatibility. There are also bigint values that can't be represented in decimal. For toString: I made it this way to have some way of getting at the input value. Other darta is publicly readable.
- Nic: if we want to expose the concept of decimal in the spec, we'd try to convert numbers to decimals, check if they're the same, throw?
- Eemeli: I think so? I think it's as Shane's presented. To have some idea for how many decimal places can be represented, we'd apply those to digit strings and bigint. If the

internal value is polymorphic, it would still be supported by Intl.NumberFormat. The opaque amount requires us to bring in restrictions. But that's it.

- Jesse: is the opaque amount too thin?
- Nic: we might have some trouble about this.
- Shane: if amount is serving primarily just the Intl case, it should perhaps go in to the Intl namespace. I'd rather have more tightly coupled with decimal, even if Intl is the main user.
- Eemeli: this goes back to the problem of what problem we're solving. Should this be in the spec? For opaque amounts: when formatting a thing, we should define the thing we're formatting separate from how we're formatting. Value is a pair or triple of number, unit/currency, maybe precision. We did see that mesure did get to stage 1. Even an opaque amount could be useful for JS users if they want to pass along values with a currency/unit. It's opaque, but at least there's the string version. So the utility is primarily in intl, but not just intl.
- Shane: namespacing: maybe it would help to talk about that more explicitly. Similar to how temporal put everything in one place. We already own the Math namespace and could use that: Math.Amount. We could use Numeric.Amount, Quant.Decimal, whatever. Does having a namespace help, thinking about coupling?
- Eemeli: not opopsed to these proposals proceeding, coupled. Keep them separate because the use cases are so different, though of course overlapping.
- Nic: if we're considering multiple classes, should it just be 2 (decimal, amount)? Or currency (amount)? Think about TypeScript.
- Eemeli: plausible to consider them separately. The list of supported currencies we currently have is explicitly listed. Any three A-Z characters are fine, for a wider set of currencies. Conversion between different currencies should not be supported. Whereas for units we may want to support various kinds of conversions. At some level, there is a lot of overlap.
- Nic: for currencies, do you have just EUR or EUR plus cents?
- Eemeli: from the Intl.NF point of view, no support for scaling a value from e.g. cents to dollars. But not inconceivable. Already Intl.NF does scaling by 100 when formatting percent values. That question should be considered.
- Shane: currencies: nice to keep them separate, but they have things they need to do together. E.g. dollars-per-mile or dollars-per-liter. EUR-per-kilogram. When you think of its that way, you question whether they're in the same space.
- Eemeli: perhaps accept "currency" as the unit, allow creation of an amount, possibly as constructor options? This probably works for most use cases but not for, say, euros per dollar.
- Nic: in plenary, we'll make clear that we have this group, fundamental questions are still under discussion. Hopefully we'll get some feedback.