April 2014 TC39 Meeting
ES6 Status and Open Issues

Allen Wirfs-Brock
Major Things in Rev23
(Language)

• For (let ;;) per iteration bindings with value propagation between iterations.
• Function variable instantiation as per Sept 2013 discussion.
• Lookahead grammar restriction to disambiguate: new super()
• Lookahead let restrictions added: IterationStatement: for (LeftHandSideExpression of AssignmentExpression)... and for (LeftHandSideExpression in Expression)...
• Reverted default for missing class constructor back to “constructor(...args) {super(...)}) because of bug 2491
• Refactored identifier syntax/semantics into IdentifierReference, BindingIdentifier, and LabelIdentifier motivated by need to allow unicode escapes in non-keyword yield identifiers
• Tweaked ordinary call to allocate non-strict mode wrapper objects using callee’s Realm
• Updated Annex B function in block legacy compatibly hack based upon Jan. meeting consensus

• Added [Yield] grammar parameter to ArrowFunction (Bug 2504)
• Added [Yield] grammar parameters for Function/Generator/Class Declarations
• Added [GeneratorParameter]] parameter to ClassExpression
• Clarified that certainly early errors don’t apply when processing parenthesized expression cover grammar bug 2506)
• 11.1.2 clarified the distinction between ES whitespace and Unicode whitespace. Added note that some Unicode white space characters are intentionally no ES whitespace
First next call to a generator

• Eliminated throw if argument is passed
• Argument is ignored and inaccessible

• Differing recollections on January discussion
• Most compelling reason: creates unnecessary difference between generator and manual implementation of equivalent iterators
Major Things in Rev23 (Library)

- Math.clz32 replaces Number.prototype.clz
- Added note that some Unicode white space characters are intentionally no ES whitespace
- Array.from({0:0,4:4, length:5}) doesn’t produce a sparse array.
- Fixed Symbol.prototype.toString Symbol.prototype.valueOf to work correctly when this value is a primitive string value
- Named %Loader%: Reflect.Loader
- Named %Realm%: ReflectRealm
- Added Reflect.Loader.prototype.@@toStringTag property
- Provide complete algorithmic definition for RegExp.prototype.replace and RegExp.prototype.search
- corrected RegExpExec so it correctly translates the match state of full Unicode RegExps back to UTF-16 capture values and endIndex.
- Documented (Annex D) fix to ES5 bug that exposed array updates to integer conversions side-effects
- Typed Array Indexing: All canonical string numeric values considered to be possible indexes rather than expando property keys, eliminated vestigial spec. language for readonly/frozen typed arrays.
- Updated Function.prototype.toMethod as per Jan. meeting.
- Updated Promises as per Jan. meeting consensu
- Switched to “ize” from secondary British “ise” spelling of “initialize” and other words.
Call for Reviewers

• Champions need to review spec. material related to their feature area

• TC39 members: please commit to reviewing specific sections
Open Issues
For of/in initialization expression scoping?

```javascript
{let x = [0,1,];
 for (let x of x) console.log(x);
}
```

- Current spec: of/in expression evaluated in enclosing scope. Log: 0 1

- Possible alternative: extra scope with uninitialized x. Throws TDZ error
Lexical scoping rules and catch parameters

• 13.14.1
try{} catch(x) {
  var x = 5
}

• Normal ES6 hoist “var” over “let” rules says this is an error.
• But, valid in ES1-5, var initializer assigns to catch parameter.
• Could only apply for destructuring catch parameters?
[[SetPrototypeOf]] circularity invariant

- Impossible to enforce if proxies exist on the prototype change.
- Eliminate the invariant
- Is there some weaker invariant we might replace it with
Eval of let/const/class

• [http://esdiscuss.org/topic/eval-of-let-etc-was-restrictions-on-let-declarations](http://esdiscuss.org/topic/eval-of-let-etc-was-restrictions-on-let-declarations)
• How to handler eval’ed lexical declaration in non-strict code
• Proposal: As if the eval was in a block and lexical declarations (except function) are scoped to the block
• Eliminates need to dynamically extend lexical scope contours.
Promise then issues:

- `p.then(42,"43")` .then(false, new Map)
  - error or default argument values if actual argument is not callable
  - If, error throw or asynch error
@@iterator for arguments object

• Own property?
• Or should be introduce an prototype object to contain it?
Web breakage: removing initializer from for-in

- http://esdiscuss.org/topic/initializer-expression-on-for-in-syntax-subject
name property of bound functions and toMethod functions?

- Currently neither have a own name property.
- Should either or both get one?
- If so, what should it be?
  - “bound foo”??
new Int32Array(iterable) ??

• Currently constructor doesn’t recognize iterables, but requires an array like.

• Need to use:
  – Int32Array.from(iterable)

• Should constructor work like Int32Array.from?
Duplicate keys when constructing Maps

• 23.1.1.2

• new Map([[“x”,1], [“x”: 2]])

• Throw or use 2 as the value of the “x” entry.
• Spec. currently says use 2, but notes that TC39 lacks consensus
Signature of Array.from map callback

• Currently, approximately:
  from(iterable, mapfn, thisArg=undefined) {
    let a = new Array;
    let index = 0;
    for (let v of iterable)
      a[index++] = mapfn.call(thisValue, v);

• Should it be:
  a[index] =
    mapfn.call(thisValue, v, index++, iterator);
Bug 1571 RegExp Syntax
ES5 changed (?=) and (!?) from zero-width atoms to assertions

• Doesn’t match web reality
• Why was this change made?
• Should we role it back?

• Also Bug 1553: Change "EscapeSequence 0 [lookahead $\notin$ DecimalDigit]" to match reality
Impl Dependencies in String.replace

• 21.1.3 String.replace substitution pattern has two “implementation defined” conditions.
• Is there a web consensus answer
RegExp toString escaping not fully specified. Why?

- 21.2.3.3.4
- The characters / or any LineTerminator occurring in the pattern shall be escaped in S as necessary to ensure that the String value formed by concatenating the Strings "/", S, "/", and F can be parsed (in an appropriate lexical context) as a RegularExpressionLiteral that behaves identically to the constructed regular expression. For example, if P is "/", then S could be "/\" or "\u002F", among other possibilities, but not "/", because /// followed by F would be parsed as a SingleLineComment rather than a RegularExpressionLiteral. If P is the empty String, this specification can be met by letting S be "(?:)".

- Why is this underspecified? Why not specify an required escaping? Do different implementation differ in their results?
To Do

• Lots of Module related cleanup and refinement.
• New eval semantics
• MOP/Proxy property enumeration API
• Cleanup completion reform and issues.
• Need to write Annex B spec. for HTML-like comments
Introduction and Language Overview

• Need ES6 paragraph for intro (Brendan?)

• Need somebody to update language overview
  – In Rev23 I added some material about classes and who they related to the prototype discussion.

• Need to recreate Annex A to reflect new grammar